CATHOLIC UNIVERSITY OF RWANDA
P.O. Box 49 BUTARE / HUYE - RWANDA
TEL: (00250) 0252 530 893 FAX: (00250) 0252 530 627
E-mail: administration@cur.ac.rw
Web: www.cur.ac.rw

FACULTY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

MODULE OF OBJECT ORIENTED PROGRAMMING USING JAVA

LEVEL TWO COMPUTER SCIENCE

, CREDITS: 20, HOURS: 100

ACADEMIC YEAR 2018-2019

LECTURER: NSENGIYUMVA Jean Marie Vianney

Master of Science in Information Systems

Email : nsengiyumva@gmail.com

Contents

PRE-REQUISITE OR CO-REQUISITE MODULES: C, C++ programming

LEARNING OUTCOMES

At the completion of the module students will be able to:

1. distinguish the terms API, IDE, and JDK
2. variables and operators
3. write a simple Java program
4. Obtain input using the JOptionPane input dialog boxes.
5. Obtain input from the console using the Scanner class.
6. Control structures
7. To implement program control with break and continue
8. To create methods, invoke methods, and pass arguments to a method
9. To use method overloading and understand ambiguous overloading.
10. (Optional) To group classes into packages
11. Arrays
12. To understand objects and classes, and use classes to model objects
13. To use UML graphical notations to describe classes and objects
14. To learn how to declare a class and how to create an object of a class
15. To understand the role of constructors when creating objects
16. To distinguish between object reference variables and primitive data type variables
17. To use classes in the Java library
18. To declare private data fields with appropriate get and set methods for data field encapsulation to make classes easy to maintain
19. To create immutable objects from immutable classes
20. To develop methods with object arguments
21. To determine the scope of variables in the context of a class
22. To use the keyword this to refer to the calling
23. To store and process objects in arrays
    (Optional GUI) To create windows using JFrame

LEARNING AND TEACHING STRATEGIES

Lectures

Practical works

Group & Individual work

Assignments

Practical works

JAVA I

Chap 1 Introduction

1 Object oriented Concepts

1.1 Object oriented Programming

Object-oriented programming (OOP) is a programming paradigm based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

The important features of object oriented programming are:

- Bottom–up approach in program design
- Programs organized around objects, grouped in classes
- Focus on data with methods to operate upon object's data
- Interaction between objects through functions
- Reusability of design through creation of new classes by adding features to existing classes

Some examples of object-oriented programming languages are C++, Java, Smalltalk, Delphi, Perl, Python, Ruby, and PHP. Grady Booch has defined object oriented programming as "a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships".

1.2 OBJECT MODEL

The object model visualizes the elements in a software application in terms of objects. In this chapter, we will look into the basic concepts and terminologies of object–oriented systems.

a) Objects and Classes

The concepts of objects and classes are intrinsically linked with each other and form the foundation of object–oriented paradigm.

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has:

- Identity that distinguishes it from other objects in the system.
- State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modeled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.

The constituents of a class are:

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

Example

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two–dimensional space. The attributes of this class can be identified as follows:

- x–coord, to denote x–coordinate of the center
- y–coord, to denote y–coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows:

- findArea(), method to calculate area
- findCircumference(), method to calculate circumference
- scale(), method to increase or decrease the radius

During instantiation, values are assigned for at least some of the attributes. If we create an object my_circle, we can assign values like x-coord : 2, y-coord : 3, and a : 4 to depict its state. Now, if the operation scale() is performed on my_circle with a scaling factor of 2, the value of the variable a will become 8. This operation brings a change in the state of my_circle, i.e., the object has exhibited certain behavior.

b) Encapsulation and Data Hiding

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.

Data Hiding

Typically, a class is designed such that its data (attributes) can be accessed only by its class methods and insulated from direct outside access. This process of insulating an object's data is called data hiding or information hiding.

Example

In the class Circle, data hiding can be incorporated by making attributes invisible from outside the class and adding two more methods to the class for accessing class data, namely:

- setValues(), method to assign values to x-coord, y-coord, and a
- getValues(), method to retrieve values of x-coord, y-coord, and a

Here the private data of the object my_circle cannot be accessed directly by any method that is not encapsulated within the class Circle. It should instead be accessed through the methods setValues() and getValues().

c) Message Passing

Any application requires a number of objects interacting in a harmonious manner. Objects in a system may communicate with each other using message passing. Suppose a system has two objects: obj1 and obj2. The object obj1 sends a message to object obj2, if obj1 wants obj2 to execute one of its methods.

The features of message passing are:

- Message passing between two objects is generally unidirectional.
- Message passing enables all interactions between objects.
- Message passing essentially involves invoking class methods.
- Objects in different processes can be involved in message passing.

d)Inheritance

Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent

classes/super-classes, and the new classes are called the derived classes/child classes/subclasses. The subclass can inherit or derive the attributes and methods of the super-class(es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an “is – a” relationship.

Example

From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow “is – a” mammal.

Types of Inheritance

Single Inheritance: A subclass derives from a single super-class.

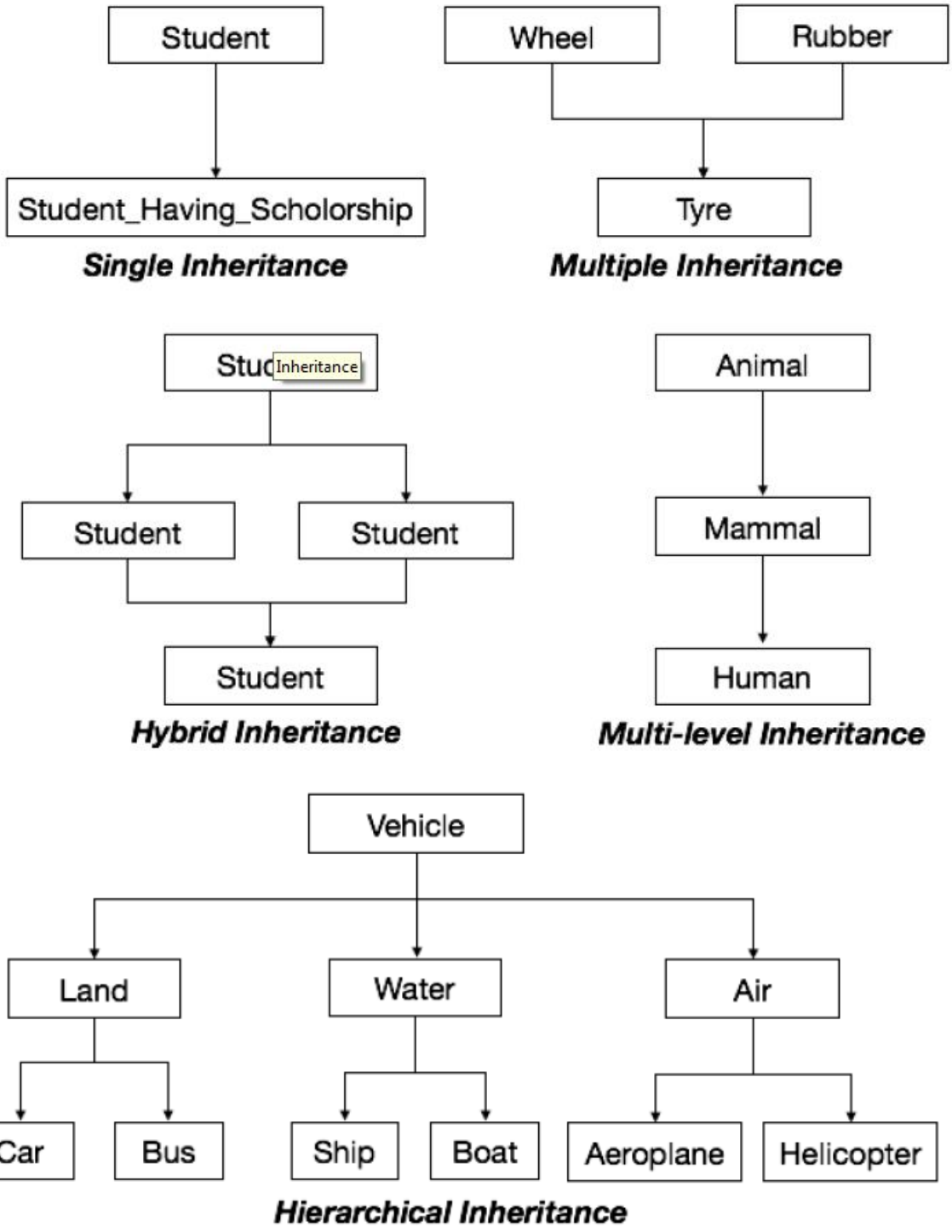
Multiple Inheritances: A subclass derives from more than one super-classes.

Multilevel Inheritance: A subclass derives from a super-class which in turn is derived from another class and so on.

Hierarchical Inheritance : A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.

Hybrid Inheritance : A combination of multiple and multilevel inheritance so as to form a lattice structure.

The following figure depicts the examples of different types of inheritance.



e) Polymorphism

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Example

Let us consider two classes, Circle and Square, each with a method findArea(). Though the name and purpose of the methods in the classes are same, the internal implementation, i.e., the procedure of calculating area is different for each class. When an object of class Circle invokes its findArea() method, the operation finds the area of the circle without any conflict with the findArea() method of the Square class.

f) Benefits of Object Model

Now that we have gone through the core concepts pertaining to object orientation, it would be worthwhile to note the advantages that this model has to offer.

The benefits of using the object model are:

- It helps in faster development of software.
- It is easy to maintain. Suppose a module develops an error, then a programmer can fix that particular module, while the other parts of the software are still up and running.
- It supports relatively hassle-free upgrades.
- It enables reuse of objects, designs, and functions.
- It reduces development risks, particularly in integration of complex systems.

1.3 OBJECT-ORIENTED SYSTEM

We know that the Object-Oriented Modeling (OOM) technique visualizes things in an application by using models organized around objects. Any software development approach goes through the following stages:

- Analysis,
- Design, and

- Implementation.

In object-oriented software engineering, the software developer identifies and organizes the application in terms of object-oriented concepts, prior to their final representation in any specific programming language or software tools.

Phases in Object-Oriented Software Development

The major phases of software development using object–oriented methodology are object-oriented analysis, object-oriented design, and object-oriented implementation.

Object Oriented Analysis

In this stage, the problem is formulated, user requirements are identified, and then a model is built based upon real–world objects. The analysis produces models on how the desired system should function and how it must be developed. The models do not include any implementation details so that it can be understood and examined by any non–technical application expert.

Object Oriented Design

Object-oriented design includes two main stages, namely, system design and object design.

System Design

In this stage, the complete architecture of the desired system is designed. The system is conceived as a set of interacting subsystems that in turn is composed of a hierarchy of interacting objects, grouped into classes. System design is done according to both the system analysis model and the proposed system architecture. Here, the emphasis is on the objects comprising the system rather than the processes in the system.

Object Design

In this phase, a design model is developed based on both the models developed in the system analysis phase and the architecture designed in the system design phase. All the classes required are identified. The designer decides whether:

- new classes are to be created from scratch,
- any existing classes can be used in their original form, or
- new classes should be inherited from the existing classes.

The associations between the identified classes are established and the hierarchies of classes are identified. Besides, the developer designs the internal details of the classes and their associations, i.e., the data structure for each attribute and the algorithms for the operations.

Object–Oriented Implementation and Testing

In this stage, the design model developed in the object design is translated into code in an appropriate programming language or software tool. The databases are created and the specific hardware requirements are ascertained. Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code.

1.4 OBJECT-ORIENTED PRINCIPLES

Principles of Object-Oriented Systems

The conceptual framework of object–oriented systems is based upon the object model. There are two categories of elements in an object-oriented system:

Major Elements : By major, it is meant that if a model does not have any one of these elements, it ceases to be object oriented. The four major elements are:

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Minor Elements : By minor, it is meant that these elements are useful, but not indispensable part of the object model. The three minor elements are:

- Typing
- Concurrency
- Persistence

Abstraction

Abstraction means to focus on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties. The essential features are relative to the context in which the object is being used.

Grady Booch has defined abstraction as follows:

“An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.”

Example : When a class Student is designed, the attributes enrolment_number, name, course, and address are included while characteristics like pulse_rate and size_of_shoe are eliminated, since they are irrelevant in the perspective of the educational institution.

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. The class has methods that provide user interfaces by which the services provided by the class may be used.

Modularity

Modularity is the process of decomposing a problem (program) into a set of modules so as to reduce the overall complexity of the problem. Booch has defined modularity as:

“Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.”

Modularity is intrinsically linked with encapsulation. Modularity can be visualized as a way of mapping encapsulated abstractions into real, physical modules having high cohesion within the modules and their inter–module interaction or coupling is low.

Hierarchy

In Grady Booch’s words, “Hierarchy is the ranking or ordering of abstraction”. Through hierarchy, a system can be made up of interrelated subsystems, which can have their own subsystems and so on until the smallest level components are reached. It uses the principle of “divide and conquer”. Hierarchy allows code reusability.

The two types of hierarchies in OOA are:

- **“IS–A” hierarchy** : It defines the hierarchical relationship in inheritance, whereby from a super-class, a number of subclasses may be derived which may again have subclasses and so on. For example, if we derive a class Rose from a class Flower, we can say that a rose “is–a” flower.

- **“PART–OF” hierarchy** : It defines the hierarchical relationship in aggregation by which a class may be composed of other classes. For example, a flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a “part–of” flower.

Typing

According to the theories of abstract data type, a type is a characterization of a set of elements. In OOP, a class is visualized as a type having properties distinct from any other types. Typing is the enforcement of the notion that an object is an instance of a single class or type. It also enforces that objects of different types may not be generally interchanged; and can be interchanged only in a very restricted manner if absolutely required to do so.

The two types of typing are:

- **Strong Typing** : Here, the operation on an object is checked at the time of compilation, as in the programming language Eiffel.

- **Weak Typing** : Here, messages may be sent to any class. The operation is checked only at the time of execution, as in the programming language Smalltalk.

Concurrency

Concurrency in operating systems allows performing multiple tasks or processes simultaneously. When a single process exists in a system, it is said that there is a single thread of control. However, most systems have multiple threads, some active, some waiting for CPU, some suspended, and some terminated. Systems with multiple CPUs inherently permit concurrent threads of control; but systems running on a single CPU use appropriate algorithms to give equitable CPU time to the threads so as to enable concurrency.

In an object-oriented environment, there are active and inactive objects. The active objects have independent threads of control that can execute concurrently with threads of other objects. The active objects synchronize with one another as well as with purely sequential objects.

Persistence

An object occupies a memory space and exists for a particular period of time. In traditional programming, the lifespan of an object was typically the lifespan of the execution of the program that created it. In files or databases, the object lifespan is longer than the duration of the process creating the object. This property by which an object continues to exist even after its creator ceases to exist is known as persistence.

2. Java Language

Java is both a programming language and a platform. Java was conceived by James Gosling @ Sun Micro Systems in the year 1991. Java programming language is designed to support the following features:

· Simple

Similar to C and C++

Easy to learn

· Object Oriented

Supports OOPS concepts

"Everything is and Object"

Advantages of Java

Java for Internet

o Easy transmitting of objects in internet as Client/Server

· Security

o Guard against malicious and supports firewall

· Portability

o Same code can be used anywhere through various platforms

· Byte code

o Byte code is highly optimized set of instructions to designed to be executed by the Java – run time system which is called JVM

Java Programs

Java can be used to be creating two types of programs:

Applets:

An applet is a Java program that can be downloaded across the network, like image, sound and video. It is a intelligent program, means that nature to change or act upon the user input.

Applications:

An application is a program that runs on your computer under the operating system of that computer.

Chap 2. How to run Java programs

1. Write your java code in a editor like notepad
2. Save the program with an extension **.java**
3. Compile the program in command mode using **javac** filename**.java**
4. Run the program in command mode using **java filename**

The first program in Java

```
/* Program for Welcome to Java */
import java.io.*;
class prog
{
```

```
public static void main(String args[ ])
{
System.out.println(“Welcome to Java”);
}
}
```

Output: Welcome to Java

In this program

· /* … */ statement is used to put comments in the program
(Comment statements will not compiled by the compiler)

· import java.io.*;
import is equivalent to include in C/C++
java.io.* ; is a package (class or library) which contains the input output classes

· class prog
class is a keyword (all the programs in java starts with class) and Prog is the class name

· public static void main(String args[ ]) the main method in java using public static and the arguments passed in to that program will be considered as Strings

· System.out.println(“Welcome to java”);
System.out.println() is a method to print the output.

Note

· Java is case sensitive, means that A≠a

· Delimiters (**;**) are the end of statements in Java

2.2 Language Basics

· **Variables**

- A variable is an item of data named by an identifier
- There are three ways of variable declarations

_ data type variable; // **declaration of a variable**

· int a;

_ data type variable=value; // **initialization of a variable**

· int a=5;

_ data type variable=expression; // **dynamic initialization**

· int a=b+c;

Operators

o An operator performs a function on one, two, or more operands.

o Operators are divided into the following categories:

Arithmetic Operators

· +, -, *, /, %

Increment and Decrement Operators

· ++, --

_Relational Operators

· >, <, >=, <=, ==, !=

_ Logical Operators

· &&, ||, !

_ Shift Operators

· <<, >>

_ Bitwise Operators

· &, |, !, ^, ~

_ Assignment Operators

· =, +=, -=, *=, ?=, %=

_ Other Operators

· ?:, [ ], ( ), (type), new

Data Types

A variable's data type determines the values that the variable can contain and the operations that can be performed on it.

There are only three groups of primitive data types:

_ Numeric types

Integer Types

- byte (1 bytes)
- short (2 bytes)
- int (4 bytes)
- long (8 bytes)

· Floating – point Types

o float (4 bytes)

o double (8 byes)

_ Character types

· char (1 byte)

_ Boolean types

· boolean (true or false)

Control Flow Statements

Looping

_ for

· for(initialization;condition;increment/decrement)

_ while

· initialization;

while(condtion)

{

Incremtment/decrement;

}

_ do .. while

· initialization;

do

{

…..

Increment/decrement);

} while (condtion);

Decision Making

_ Simple if

· if (condtion)

{

true block statements;

}

_ if…else

· if (condtion)

{

true block statements;

}

else

{

false block statements;

}

Nested if’s

· if (condtion)

{

true block statements;

}

else if

{

true block statements;

}

else

{

false block statements;

}

Selection

_ switch case

· switch(choice)

{

case caselabel:

statements;

break;

case caselabel:

statements;

```
break;
……
default:
default statements;
}
```

Branching

_ break

· break;

_ continue

· continue;

2.3 Simple Programs

1. /* Program for simple addition */

```
import java.io.*;
class prog1
{
public static void main(String args[ ])
{
int a,b,c;
a=5;
b=10;
c=a+b;
System.out.println(“C=”+c);
}
}
```

2. /* Program to find greatest number */

```
import java.io.*;
class prog2
{
public static void main(String args[ ])

{
```

```
int a,b;
a=5;
b=10;
if (a>b)
System.out.println(“A is Greater”);
else
System.out.println(“B is Greater);
}
}
```

```
/* Program to print 1 to 10 using for loop*/
import java.io.*;
class prog3
{
public static void main(String args[ ])
{
for(int i=1;i<=10;i++)
System.out.println(“i”);
}
}
```

4.

```
/* Program to print 1 to 10 using while loop*/
import java.io.*;
class prog4
{
public static void main(String args[ ])
{
int i=1;
while(i<=10)
{
System.out.println(“i”);
i++;
}
```

```
}
}
```

```
/* Program to print 1 to 10 using do..while loop*/
import java.io.*;
class prog5
{
public static void main(String args[ ])
{
int i=1;
do
{
System.out.println(“i”);
i++;
} while(i<=10);
}
}
```

6.

```
/* Program to find true or false*/
import java.io.*;
class prog6
{
public static void main(String args[ ])
{

int a=5, b=10;
System.out.println(a>b);
}
}
```

7.

```
/* Program for type conversion*/
import java.io.*;
class prog7
{
```

```
public static void main(String args[ ])
{
int a=5, b;
double c=353.769, d;
d=a;
System.out.println("Int to Double(Automatic)="+ d);
b=(int) c;
System.out.println("Double to Int(External)="+b);
}
}
```

Chap3. User Defined Classes

3.1 How to design user defined classes

Class

A class is a collection of objects. A description of a group of objects with same properties. There are two types of elements of a class:

· A data member is an object of any type

· Member functions (methods) to access or operate on those members

Defining Classes

A class definition is always introduced by the reserved word **class**. The name of the class is indicated after the word **class**.

· The data or variables defined within a class are called instance variables.

· The methods and variables defines within a class are called members.

A Simple Class

A class called Box that defines three instance variables: width, height and depth.

```
class Box // defining class
{
double width; //
double height; // // instance variables
double depth; //
}
```

Objects

Objects are the runtime or real time entities. Each object has its own memory.

Each object has a type.

type name= class name();

Declaring Objects

The **new** operator dynamically allocates (that allocates at run time) memory for an object. Java supports the objects which are created by **new**, to destroy automatically and free the memory.

To declare an object of type Box:

Box mybox=new Box(); // allocate a Box object

The first class program

```
/* Program for class Box */
import java.io.*;
class Box
{
int width;
int height;
int depth;
}
class BoxProgram
{
public static void main (String args[ ])
{
Box mybox=new Box();
mybox.width=5;
mybox.height=10;
mybox.depth=15;
System.out.println("Width="+mybox.width);
System.out.println("Height="+mybox.height);
System.out.println("Depth="+mybox.depth);
```

```
}
}
```

Output:

Width=5

Height=10

Depth=15

3.2 Methods

Instead of functions, in Java it's known as methods. A method can be called only for an object and that objects must able to perform that method call. The methods of an object called by following a (.) dot

objectname.method();

Adding method to the Box class

Methods can be defined in the class itself. A method can be called any number of times. The following program adds a method to the Box class to print the output.

```
/* Program for class Box with methods */
import java.io.*;
class Box

{
int width;
int height;
int depth;
void show()
{
System.out.println("Width="+mybox.width);
System.out.println("Height="+mybox.height);
System.out.println("Depth="+mybox.depth);
}
}
class BoxMethods
{
```

```
public static void main (String args[ ])
{
Box mybox=new Box();
mybox.width=5;
mybox.height=10;
mybox.depth=15;
mybox.show();
}
}
```

Output:

```
Width=5
Height=10
Depth=15
```

Method with Parameters

Methods can be defined along with the parameters. Parameters allow a method to pass the values from the calling method. The following program adds a method that takes parameters.

```
/* Program for class Box with methods and parameters */
import java.io.*;
class Box
{
int width;
int height;
int depth;
void show()
{
System.out.println("Width="+mybox.width);

System.out.println("Height="+mybox.height);
System.out.println("Depth="+mybox.depth);
}
```

```
void setvalue(int w, int h, int d)
{
width=w;
height=h;
depth=d;
}
}
class BoxParams
{
public static void main (String args[ ])
{
Box mybox=new Box();
mybopx.setvalue(1,2,3);
mybox.show();
mybox.setvalue(5,10,15);
mybox.show();
}
}
```

Output:

```
Width=1
Height=2
Depth=3
Width=5
Height=10
Depth=15
```

Overloading Methods

Defining two or more methods within the same class that shares the same name as long as their parameter declarations are different. The following program uses method overloading to overload three types of parameter sets.

```
/* Program for class Box with method overloading */
```

```
import java.io.*;
class Box
{
int width;
int height;
int depth;

void show()
{
System.out.println(“Width=”+mybox.width);
System.out.println(“Height=”+mybox.height);
System.out.println(“Depth=”+mybox.depth);
}
void setvalue(int w)
{
width=height=depth=w;
}
void setvalue(int w, int h)
{
width=w;
height=h;
depth=30;
}
void setvalue(int w, int h, int d)
{
width=w;
height=h;
depth=d;
}
}

class BoxMethodOverload
```

```
{
public static void main (String args[ ])
{
Box mybox=new Box();
mybopx.setvalue(10);
mybox.show();
mybox.setvalue(10,20);
mybox.show();
mybox.setvalue(100,200,300);
mybox.show();
}
}
```

Output:

```
Width=10
Height=10
Depth=10

Width=10
Height=20
Depth=30
Width=100
Height=200
Depth=300
```

3.3 Constructors

Constructors are used to initialize all the variables in the class, immediately upon the creation of object. It has the same name as class. Constructors have no type, but it can take parameters.

```
classname()
{
// constructing the class
}
```

The following program uses a constructor to initialize the dimensions of class Box

```
/* Program for class Box with constructors */
import java.io.*;
class Box
{
int width;
int height;
int depth;
void show()

{
System.out.println(“Width=”+mybox.width);
System.out.println(“Height=”+mybox.height);
System.out.println(“Depth=”+mybox.depth);
}
Box()
{
width=10;
height=20;
depth=30;
}

}
class BoxCons
{
public static void main (String args[ ])
{
Box mybox=new Box();
mybox.show();
}
}
```

Output:

Width =10

Height=20

Depth=30

Parameterized Constructors

Parameterized constructors are used to set the different values of class members which is specified by those parameters. The following program shows how parameterized parameters are used to set the dimension of class Box

```
/* Program for class Box with constructors and parameters */
import java.io.*;
class Box
{
int width;
int height;
int depth;
void show()

{
System.out.println("Width="+mybox.width);
System.out.println("Height="+mybox.height);
System.out.println("Depth="+mybox.depth);
}
Box( int w, int h, int d)
{
width=w;
height=h;
depth=d;
}
}
class BoxCons

{
public static void main (String args[ ])
```

```
{
Box mybox=new Box(1,2,3);
Box yourbox=new Box(10,20,30);
mybox.show();
yourbox.show();
}
}
```

Output:

```
Width =1
Height=2
Depth=3
Width =10
Height=20
Depth=30
```

Overloading Constructors

The same as method overloading, constructors also overloaded with different set of parameters. The following program shows the constructor overloading.

```
/* Program for class Box with constructors overloading */
import java.io.*;
class Box
{
int width;
int height;
int depth;
void show()

{
System.out.println("Width="+mybox.width);
System.out.println("Height="+mybox.height);
System.out.println("Depth="+mybox.depth);
}
```

```
Box()
{
width=10;
height=10;
depth=10;
}
Box(int w)
{
width=height=depth=w;
}
Box( int w, int h)
{
width=w;
height=h;
depth=100;
}
Box( int w, int h, int d)
{
width=w;
height=h;
depth=d;
}
}
class BoxCons
{
public static void main (String args[ ])
{
Box mybox1=new Box();
Box mybox2=new Box(50);
Box mybox3=new Box(100,100);
Box mybox4=new Box(100,200,300);
```

```
mybox1.show();
mybox2.show();
mybox3.show();
mybox4.show();
}
}
```

Output:

```
Width =10
Height=10
Depth=10

Width =50
Height=50
Depth=50

Width =100
Height=100
Depth=100

Width =100
Height=200
Depth=300
```

Chap4. Static Methods and Variables

4.1 Understanding Static Methods and Variables

It is possible to create a member that can be used by itself. To create such a member **static** is used. When a member is declared as static, it can be accessed before any objects of its class are created.

The most common example of a static member is **main ()**. main() is declared as static because it must be called before any objects exists.

Variables declared as static are like global variables. When objects of class are

declared, the instances of the class share the same name static variable.

The following program shows how static variables and methods are declared.

```
// demonstrate static variables, methods, and blocks.
import java.io.*;
class UseStatic
{
static int a = 10;
static int b;
static void show()
{
System.out.println("a = " + a);
System.out.println("b = " + b);
}
static
{
System.out.println("Static block initialized.");
b = 20;
}
public static void main(String args[])
{

show();
}
}
```

Output:

a=10

b=20

As soon as the UseStatic class is loaded, all the static statements are run. First a is set to 10, then the static block executes, and b is set to 20. Then main() is called, which calls the method show(), it prints the value for a and b.

4.2 Using Static outside of the class

· To call a static method from outside its class, classname.method() can be used.

· To call a static member from outside its class, classname.variable can be used.

The following program shows how the static method and static variables called outside of their class.

```
//demonstrate static methods and variables called outside from the class
import java.io.*;
class StaticDemo
{
static int a = 33;
static int b = 99;
static void callme()
{
System.out.println("a = " + a);
}
}

class StaticProg
{
public static void main(String args[])
{
StaticDemo.callme();
System.out.println("b = " + StaticDemo.b);
}
}
```

Output:

a=33

b=99

Chap5. Characters and Strings

5.1 Characters

The data type used to store characters is **char**. In Java the range of char is 0 to 65,536. The standard set of characters known as ASCII ranges 0-127 and extended character set ranges from 0 – 255.

A simple program for char variables:

```
/* program for char demo*/
import java.io.*;
class chardemo
{
public static void main(String args[])
{
char c=0;
for(int i=0;i<=255;i++)
{
System.out.println("ASCII value for "+c+" is "+(int) c);
c++;
}
}
}
```

This program will print the ASCII values from 0 – 255.

Static Methods of Character

Character includes several static methods that categorize characters and alter their case. The following program demonstrates these methods.

```
// Demonstrate several Is... methods.
import java.io.*;
class IsDemo
{
public static void main(String args[])

{
```

```
char a[] = {'a', 'b', '5', '?', 'A', ' '};
for(int i=0; i<a.length; i++) {
if(Character.isDigit(a[i]))
System.out.println(a[i] + " is a digit.");
if(Character.isLetter(a[i]))
System.out.println(a[i] + " is a letter.");
if(Character.isWhitespace(a[i]))
System.out.println(a[i] + " is whitespace.");

if(Character.isUpperCase(a[i]))
System.out.println(a[i] + " is uppercase.");
if(Character.isLowerCase(a[i]))
System.out.println(a[i] + " is lowercase.");
}
}
}
```

Output:

```
a is letter
a is lowercase
b is letter
b is lowercase
5 is digit
A is a letter
A is a uppercase
is a whitespace
```

5.2 Strings

In Java, a string is a sequence of characters. Java implements strings as object of type String rather than character arrays. StringBuffer is a companion class to String, whose objects contain Strings that can be modified.

Sting and StringBuffer classes are defined in java.lang. Thus they are available to all programs automatically.

String Constructors

The string class supports several constructors. To create an empty string **new** operator is used.

String S=new String();

String Length

The length of a string is the number of characters that it contains. To obtain this value the int length() method is used.

char c[]={'a','b','c'};

String s=new String(c);

System.out.println(s.lenght());

The above fragment prints 3.

String Concatenation

Java allows to chain together a series of + operations. The following fragment concatenates three strings.

String s1=" Welcomes";

String s2="NUR"+s1+" you";

System.out.println(s2);

This displayes the strings "NUR welcomes you"

Using concat()

To concatenate two strings we can use concat()

String s1="one";

String s2=s1.concat("two");

This will puts the string "onetwo"

String Comparison

· **equals()**

To compare two strings for equality, we can use equals()

String s1="hai";

String s2="hello";

String s3="hai";

System.out.println(s1.equals(s2));

System.out.println(s1.equals(s3));

The output for the above fragment will be

false //where s1 and s2 not equals

true //where s1 and s3 equals

· **equalsIgnoreCase()**

To compare two strings that ignores case differences, we can use equalsIgnoreCase()

String s1=”hai”;

String s2=”HAI”;

System.out.println(s1.equals(s2));

System.out.println(s1.equalsIgnoreCase(s2));

The output for the above fragment will be

false //where s1 and s2 not equals (considering case)

true //where s1 and s2 equals (ignoring case)

Searching Strings

The string class provides two methods that allow you to search the specified character or substring.

· indexOf()

Searches the first occurrence of the character or substring.

· lastIndexOf()

Searches the last occurrence of the character or substring.

String S=”Welcome to the String World in side the Java”;

System.out.println(s.indexOf(‘t’));

System.out.println(s.indexOf(“the”));

System.out.println(s.lastIndexOf(‘t’));

System.out.println(s.lastIndexOf(“the”));

The above fragment will display

8 // ‘t’ starts at 8th position

14 // “the” starts at 14th position

36 // ‘t’ finishes at 36th position

36 //”the” finishes at 36th position

Modifying String

· replace()

o The replace() method replaces all occurrences of one character to a specified one.

String s=”Hello”.replace(‘l’,’w’);

Puts the string “Hewwo” into s.

trim()

o The trim() method returns a copy of string by eliminating leading and trailing whitespaces.

String s=” Hello World “.trim();

Puts the string “Hello World” into s.

Changing Case

· The method toLowerCase() converts all the characters in a sting to lower Case

String s=”HELLO WELCOME”;

System.out.println(s.toLowerCase());

This fragment will display “hello welcome”.

The method toUpperCase() converts all the characters in a sting to upper case

String s=” hello welcome”;

System.out.println(s.toUpperCase());

This fragment will display “HELLO WELCOME”.

String Buffer

StingBuffer is a peer class of String that provides much of the functionality of strings.

insert()

o The insert() method inserts one string into another

StringBuffer sb=new StringBuffer(“I Java”);

sb.insert(2,” like ”);

System.out.println(sb);

This fragment will display “I like Java”.

delete() and deleteCharAt()

o The delete() method deletes a sequence of characters

StringBuffer sb=new StringBuffer(“Welcome to Strings”)

sb.delete(8,10);

System.out.println(sb);

This fragment will display “Welcome Strings”.

o The deletCharAt() method deletes the character by specifying the index

StringBuffer sb=new StringBuffer(“Welcome to Strings”)

sb.deleteCharAt(6);

System.out.println(sb);

This fragment will display “Welcome Strings”.

reverse()

o The reverse() method reverse the characters within the string

StringBuffer sb=new StringBuffer(“Welcome”)

sb.reverse();

System.out.println(sb);

This fragment will display “emocleW”.

Chap 6. One – Dimensional Arrays

6.1 Arrays of static allocation declaration

An one-dimensional array is a list of same type of variables. The general form of one-dimensional array declaration is

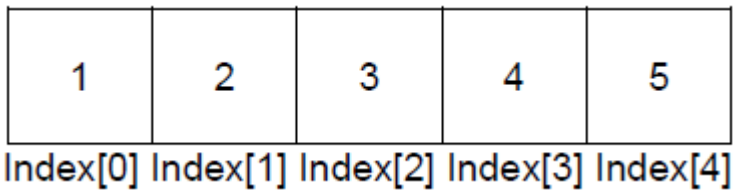
type array_name[ ]={values};

Here the type declares the basic type of array. The base type determines the data type of each element that comprises the array. The following example declares an array named “intarray” with the type of int.

int intarray[ ]={1,2,3,4,5);

In array, the indexes will be start from 0 and end with (array.length-1). Say for example, to retrieve the value 3, the subscript used is

Conceptual Diagram for 1D array



Sytem.out.println(intarray[2]);

The above output statement will print 3.

The following program specifies how 1D array are declared with initialization and access those array elements.

```
/* program for 1D Arrays */
import java.io.*;
class oneDarray
{
public static void main(String args[ ])
{
int intarray[ ]={1,2,3,4,5};
Sytem.out.println("First Element ="+intarray[0]);
Sytem.out.println("Second Element ="+intarray[1]);
Sytem.out.println("Third Element ="+intarray[2]);
Sytem.out.println("Fourth Element ="+intarray[3]);
Sytem.out.println("Fifth Element ="+intarray[4]);
}
}
```

Output:

```
1
2
3
```

4

5

Here is one more example that uses one-dimensional arrays

```
/* Program to calculate average using arrays*/
import java.io.*;
class Average
{
public static void main(String args[])
{
double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};
double result = 0;
int i;
for(i=0; i<5; i++)
result = result + nums[i];
System.out.println("Average is " + result / 5);
}
}
```

Output:

12.3

6.2 Arrays of dynamic allocation declaration

- Dynamic allocation of arrays will be declared by using the **new** keyword.
- The elements in the array allocated by new will automatically be initialized to zero.

```
int intarray[ ]=new int[5];
```

The size of the array can be dynamically received by user input also.

For example

```
int a;
```

a will be assigned by some value from the user input

now

```
int intarray[]=new int[a];
```

This statement will allocate the array of a size a.

The following program shows how the values been set the array elements

```
// Demonstrate a one-dimensional array with new.
import java.io.*;
class Array
{
public static void main(String args[])
{
int month_days[];
month_days = new int[12];
month_days[0] = 31;
month_days[1] = 28;
month_days[2] = 31;
month_days[3] = 30;
month_days[4] = 31;
month_days[5] = 30;
month_days[6] = 31;
month_days[7] = 31;
month_days[8] = 30;
month_days[9] = 31;
month_days[10] = 30;
month_days[11] = 31;
System.out.println("April has " + month_days[3] + " days.");
}
}
```

Output:

April has 30 days.

To get the input from the user

Java is not directly supporting to input the number values by the user. To perform this it has some indirect methods. The following steps can be made to input a number value from the user.

1. We can use DataInputStream or BufferedReader to get the input.
2. The input values will assign to String variables
3. Using parse method we can convert the strings to required number values

The following program shows how to read the input from the user

```
// Demonstrate a one-dimensional array with user input.
import java.io.*;
class ArrayInput
{
public static void main(String args[])
{
int n;
String s;
DataInputStream ds=new DataInputStream(System.in);
System.out.println("Enter the size of the Array=");
s=ds.readLine();
n=Integer.parseInt(s);
int a[]=new int[n];
System.out.println("Enter the elements of the Array");
for(int i=0;i<a.length;i++)
{
s=ds.readLine();
a[i]=Integer.parseInt(s);
}
System.out.println("The elements of the Array");
for(int i=0;i<a.length;i++)
System.out.println(a[i]);
}
}
```

The above program will get the input from the user and according to user specified size the array will be declared dynamically. According to the values specified by the user, the array will hold the elements and display the elements.

The same program can be written by using BufferedReader

int a;

String s;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in);

System.out.println("Enter the size of the Array=");

s=br.readLine();

a=Integer.parseInt(s);

The above fragment will puts the input value which is entered by the user, into the variable a.

chap7. Multi - Dimensional Arrays

7.1 Arrays of Arrays

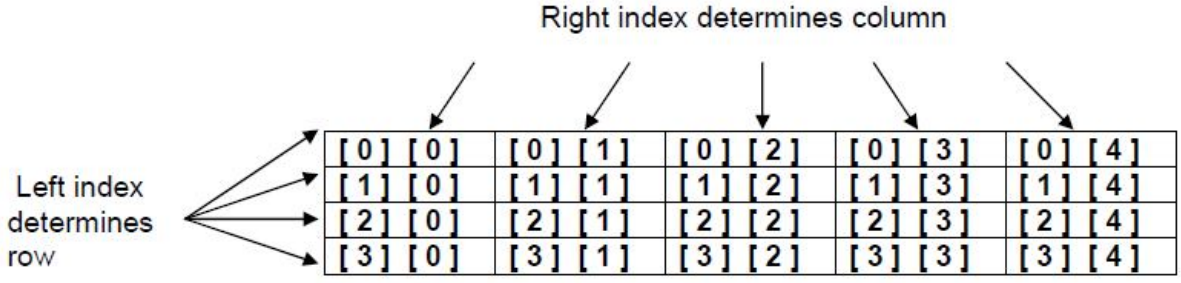
More than one dimensional, the arrays are called as multi-dimensional arrays. In Java multi-dimensional arrays are actually arrays of arrays.

The following declaration states the two dimensional array

int twoD[ ] [ ]=new int[4] [5];

4 represent the row and 5 represent the column of the array.

Conceptual Diagram for two dimensional arrays



The following program numbers each element in the array from left to right, top to bottom and displays the values.

```
// Demonstrate a two-dimensional array.
import java.io.*;
class TwoDArray
{
public static void main(String args[])
```

```
{
int twoD[][]= new int[4][5];
int i, j, k = 0;
for(i=0; i<4; i++)
for(j=0; j<5; j++) {
twoD[i][j] = k;
k++;
}

for(i=0; i<4; i++) {
for(j=0; j<5; j++)
System.out.print(twoD[i][j] + " ");
System.out.println();
}

}
}
```

Output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

7.2 Initializing two dimensional arrays

The two dimensional arrays are initialized like one dimensional array with a curly braces. But here, we use curly braces inside curly braces.

```
int aa[][]={
{ 1,2,3},
{4,5,6},
{7,8,9}
};
```

The above declaration initializes the two dimensional arrays with a size of 3x3.

Two dimensional arrays can be initialized with some arithmetic operation like the

following program has.

```
// Initialize a two-dimensional array.
class Matrix {
public static void main(String args[]) {
double m[][] = {
{ 0*0, 1*0, 2*0, 3*0 },
{ 0*1, 1*1, 2*1, 3*1 },
{ 0*2, 1*2, 2*2, 3*2 },
{ 0*3, 1*3, 2*3, 3*3 }
};
int i, j;
for(i=0; i<4; i++) {
for(j=0; j<4; j++)

System.out.print(m[i][j] + " ");
System.out.println();
}
}
}
```

Output:

```
0.0 0.0 0.0 0.0
0.0 1.0 2.0 3.0
0.0 2.0 4.0 6.0
0.0 3.0 6.0 9.0
```

Chap 8. Algorithms for Sorting and Searching Arrays

8.1 Sorting Arrays

Sorting is the process of putting data in order; either numerically or alphabetically. There are literally hundreds of different ways to sort arrays. The basic goal of each of these methods is the same: to compare each array element to another array element and swap them if they are in the wrong position.

The selection sort

In selection sort during each iteration, the unsorted element with the smallest (or largest) value is moved to its proper position in the array.

The number of times the sort passes through the array is one less than the number of items in the array. In the selection sort, the inner loop finds the next smallest (or largest) value and the outer loop places that value into its proper location.

The following table shows the swap of elements in each iteration. Here each iteration includes the combination of both loops.

Selection Sort (Descending Array)						
Array at beginning:	84	69	76	86	94	91
After iteration 1:	84	91	76	86	94	69
After iteration 2:	84	91	94	86	76	69
After iteration 3:	86	91	94	84	76	69
After iteration 4:	94	91	86	84	76	69
After iteration 5:	94	91	86	84	76	69

The algorithm for selection sort:

```
int [ ] selection_sort(int [ ]array)
{
int i, j, first, temp;
int array_size = array.length( );
for (i=0; i<array.lenth; i++)
{
for (j=i+1; j<a.length; j++) //Find smallest element between the positions 1 and i.
{
if (array[j] < array[j])
temp = array[i]; // Swap smallest element found with one in position i.
array[j] = array[j];
array[j] = temp;
}
```

```
}

return array;
}
```

The following program shows how selection sort can be implemented using class.

```
//demonstration of selection sort
import java.io.*;
class selection
{
int i,j,temp=0;
int []sortt(int []a)
{
for(i=0;i<a.length;i++)
for(j=i+1;j<a.length;j++)
if(a[i]<a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
return a;
}
}
class sorting
{
public static void main(String args[])
{
int x[]={84,69,76,86,94,91};
int y[]=new int[x.length];
selection sl=new selection();
y=sl.sortt(x);
for(int s=0;s<y.length;s++)
```

```
System.out.println(y[s]);
}
}
```

Output:

94

91

86

84

76

69

8.2 Searching Arrays

A fast way to search a sorted array is to use a binary search. The idea is to look at the element in the middle. If the key is equal to that, the search is finished. If the key is less than the middle element, do a binary search on the first half. If it's greater, do a binary search of the second half.

The algorithm for binary search:

```
int AL (int[] array, int seek)
{
int bot = -1;
int top = size;
while (top - bot > 1) {
int mid = (top + bot)/2;
if (array[mid] < seek) bot = mid;
else /* array[mid] >= seek */ top = mid;
}
return top;
}
```

The implementation of the above algorithm is implemented using class.

```
//demosntration of binarch algorithm
import java.io.*;
```

```
class bsearch
{
int search (int[] array, int seek)
{
int bot = -1;
int top = array[array.length-1];
while (top - bot > 1)
{
int mid = (top + bot)/2;
if (array[mid] < seek) bot = mid;
else top = mid;
}
return top;
}

class binarysearch
{
public static void main(String args[])
{
int x[]={1,2,3,4,5};
int n=3;
bsearch bs=new bsearch();

int y=bs.search(x,n);
System.out.println("The index of array element "+n+" = "+y);
}
}
```

Output:

The index of array element 3 = 2.

Chap9. Composition of Classes

Composition of Classes

Defining a class within another class is known as nested classes or inner classes or composition classes.

The following program illustrates how to define an inner class.

```
// demonstrate an inner class.
class Outer
{
int outer_x = 100;
void test() {
Inner inner = new Inner();
inner.display();
}
// this is an innner class
class Inner
{
void display() {
System.out.println("display: outer_x = " + outer_x);
}
}
}

class InnerClassDemo

{
public static void main(String args[])
{
Outer outer = new Outer();
outer.test();
}
}
```

Output:

display: outer_x=100

Chap10. Inheritance

Inheritance

Inheritance is used to inherit the properties from one class to another. The class that is inherited is called a super class. The class that does inheriting is called a sub class.

The following program creates a super class called A and a subclass B. The keyword **extends** is used to create a sub class of A.

```
// A simple example of inheritance.
// Create a superclass.
class A
{
int i, j;
void showij() {
System.out.println("i and j: " + i + " " + j);
}
}
// Create a subclass by extending class A.
class B extends A
{
int k;
void showk() {
System.out.println("k: " + k);
}
void sum() {
System.out.println("i+j+k: " + (i+j+k));
}
}
class SimpleInheritance
{
```

```
public static void main(String args[])

{
A superOb = new A();
B subOb = new B();
// The superclass may be used by itself.
superOb.i = 10;
superOb.j = 20;
System.out.println("Contents of superOb: ");
superOb.showij();

System.out.println();

/* The subclass has access to all public members of
its superclass. */
subOb.i = 7;
subOb.j = 8;
subOb.k = 9;
System.out.println("Contents of subOb: ");
subOb.showij();
subOb.showk();
System.out.println();
System.out.println("Sum of i, j and k in subOb:");
subOb.sum();
}
}
```

Output:

```
Contents of SuperOb:
i and j: 10 20
Contents of SubOb:
i and j: 7 8
k: 9
Sum of I, j and k in subOb:
```

i+j+k: 24

Chap11. Recursion

Recursion

Recursion is the process of defining something in terms of itself. A method that calls itself is said to be recursive.

The classical example of recursion is the computation of the factorial number.
The factorial of a number N is the product of all the whole numbers between 1 and N.
for example, 3 factorial is 1x2x3, or 6.
The following program illustrates the recursion.

```
// A simple example of recursion.
class Factorial
{
// this is a recusive function
int fact(int n) {
int result;

if(n==1) return 1;
result = fact(n-1) * n;
return result;
}
}
class Recursion
{
public static void main(String args[])
{
Factorial f = new Factorial();
System.out.println("Factorial of 3 is " + f.fact(3));
System.out.println("Factorial of 4 is " + f.fact(4));
System.out.println("Factorial of 5 is " + f.fact(5));
}
```

}

Output:

Factorial of 3 is 6

Factorial of 4 is 24

Factorial of 5 is 120

JAVA II

chap1. Introduction to GUI Programming

1.1 Applet Basics

Applets are small applications that are accesses on an Internet server, transported over the internet, automatically installed, and run as part of a web document.

A simple applet program

```
// Program for simple applet
import java.awt.*;
import java.applet.*;
public class SimpleApplet extends Applet
{
public void paint(Graphics g)
{
g.drawString("A Simple Applet", 20, 20);
}
}
```

In this program:

• The applet begins with two import statements.

• Applets interact with the user through AWT (Abstract Window Tool kit) not by console I/O classes

• Every applet program must be the sub class of Applet package.

• The class must be declared as publics, because it will be accessed by code the code that is outside the program.

• All the applet programs inherited from Applet.

• The paint method is used draw or redraws the output in an applet window.

• Graphics is one of the parameter of paint method.

• The graphics method outputs a string at the specified locations X, Y

To run applet

1. Use any Java compatible web browser

2. Use Applet viewer (a tool which comes along withSDK)

How to run applet programs

There are two ways to run the applet programs

1. We can have an individual applet code and an individual html code which calls the applet class file.

2. We can have a direct code which contains both applet and html code. (The html code will not be considered while the time of java compilation).

By using the first method to run the above program:

1. Compile SimpleApplet.java

2. Write a HTML file which can call the SimpleApplet.class and specify the size of the applet window you want.

A simple html code which can call the applet

```
<HTML>
<HEAD>
<TITLE>HELLO APPLET</TITLE>
</HEAD>
<BODY>
<APPLET CODE=”SimpleApplet.class” WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

3. Save the html file and open from any java compatible web browser. (or)

4. You can use appletviewer from your command line to execute this html file

By using the direct method to run the above program

The same program can be added with few lines of html code (which should mark as comments).

```
// Program for simple applet using direct method
import java.awt.*;
import java.applet.*;
/*
<applet code=”SimpleApplet.class” width=150 height=25>
```

```
</applet>
*/
public class SimpleApplet extends Applet
{
public void paint(Graphics g)
{
g.drawString("A Simple Applet", 20, 20);
}
}
```

Output:

The output for these two methods gives you the same result. When you run this program you can see a applet window which prints you string “A Simple Applet” in the location of x=20 and y=20 in that window.

Applet Initialization and Termination

There are some methods provides the basic mechanism by which the browser or applet viewer interfaces to the applet and controlsits execution. They are,

1. init()

The init() method is the first method to be called.

2. start()

The start() method is called after init().

3. paint()

The paint() method is called each time your applet’s output redrawn.

4. stop()

The stop() method is called when a web browser leaves the HTML document containing the applet

5. destroy()

The destroy() method called when your applet needs to be removed completely form memory.

In which the paint() method belongs to the AWT component class. All other four methods defined by Applet.

The following program is used to demonstrate this applet skeleton:

```
// simple applet that sets the foreground and background colors and outputs a string.
import java.awt.*;
import java.applet.*;
/*
<applet code="Sample" width=300 height=50>
</applet>
*/
public class Sample extends Applet
{
String msg;
public void init()
{
setBackground(Color.cyan);
setForeground(Color.red);
msg = "Inside init( ) --";
}
public void start()
{
msg += " Inside start( ) --";
}
public void paint(Graphics g)
{
msg += " Inside paint( ).";
g.drawString(msg, 10, 30);
}
}
```

Applets with status window

showstaus() method is used to display the message about what is occurring in the applet. The following program demonstrates showstatus().

```
// Using the Status Window.
```

```
import java.awt.*;
import java.applet.*;
/*
<applet code="StatusWindow" width=300 height=50>
</applet>
*/
public class StatusWindow extends Applet
{
public void init()
{
setBackground(Color.cyan);
}
public void paint(Graphics g)
{
g.drawString("This is in the applet window.", 10, 20);
showStatus("This is shown in the status window.");
}
}
```

1.2 Graphics in Applet

The graphics class defines a number of drawing functions. Each shape can be drawn edge-only or filled. Objects are drawn and filled in the currently selected graphics color, which is black by default.

Drawing lines: using drawLine(int startX, int startY,int endX, int endY)

```
// Draw lines
import java.awt.*;
import java.applet.*;
/*
<applet code="Lines" width=300 height=200>
</applet>
*/
public class Lines extends Applet
```

```
{
public void paint(Graphics g)
{
g.drawLine(0, 0, 100, 100);
g.drawLine(0, 100, 100, 0);
g.drawLine(40, 25, 250, 180);
g.drawLine(75, 90, 400, 400);
g.drawLine(20, 150, 400, 40);
g.drawLine(5, 290, 80, 19);
}
}
```

Drawing Rectangles: using 1. drawRect(int top, int left, int width, int height)

2. fillRect(int top, int left, int width, int height)

```
// Draw rectangles
import java.awt.*;
import java.applet.*;
/*
<applet code="Rectangles" width=300 height=200>
</applet>
*/
public class Rectangles extends Applet
{
public void paint(Graphics g)
{
g.drawRect(10, 10, 60, 50);
g.fillRect(100, 10, 60, 50);
g.drawRoundRect(190, 10, 60, 50, 15, 15);
g.fillRoundRect(70, 90, 140, 100, 30, 40);
}
}
```

Drawing Ellipses: using 1. drawOval(int top, int left, int width, int height)

2. fillOval(int top, int left, int width, int height)

```
// Draw Ellipses
import java.awt.*;
import java.applet.*;
/*
<applet code="Ellipses" width=300 height=200>
</applet>
*/
public class Ellipses extends Applet
{
public void paint(Graphics g)
{
g.drawOval(10, 10, 50, 50);
g.fillOval(100, 10, 75, 50);
g.drawOval(190, 10, 90, 30);
g.fillOval(70, 90, 140, 100);
}
}
```

Drawing Arcs: using

1. drawArc(int top, int left, int width, int height, int startangle, int sweepangle)
2. fillArc(int top, int left, int width, int height, int startangle, int sweepangle)

```
// Draw Arcs
import java.awt.*;
import java.applet.*;
/*
<applet code="Arcs" width=300 height=200>
</applet>
*/
public class Arcs extends Applet
{
public void paint(Graphics g)
```

```
{
g.drawArc(10, 40, 70, 70, 0, 75);
g.fillArc(100, 40, 70, 70, 0, 75);
g.drawArc(10, 100, 70, 80, 0, 175);
g.fillArc(100, 100, 70, 90, 0, 270);
g.drawArc(200, 80, 80, 80, 0, 180);
}
}
```

Working with Color

Color defines to several constants to specify a number of common colors.

• To set the background color of an applet's window use setBackground( )

• To set the foreground color of an applet's window use setForeground( )

Syntax:

```
setBacground(Color newColor)
setForeground(Color newColor)
```

Here, newColor specifies the new color as shown asbelow.

Color.black Color.megenta

Color.blue Color.orange

Color.cyan Color.pink

Color.green Color.red

Color.gray Color.white

Color.lightGray Color.yellow

Color.darkGray

There is one other form for Color constructor is:

Syntax:

```
Color(int red, int green, int blue)
```

This constructor takes three integers that specifythe color as a mix of red, green and blue. These values must be between 0 and 255.

```
// Demonstrate color.
import java.awt.*;
import java.applet.*;
```

```
/*
<applet code="ColorDemo" width=300 height=200>
</applet>
*/
public class ColorDemo extends Applet
{
public void paint(Graphics g)
{
Color c1 = new Color(255, 100, 100);
Color c2 = new Color(100, 255, 100);
Color c3 = new Color(100, 100, 255);
g.setColor(c1);
g.drawLine(0, 0, 100, 100);
g.drawLine(0, 100, 100, 0);
g.setColor(c2);
g.drawLine(40, 25, 250, 180);
g.drawLine(75, 90, 400, 400);
g.setColor(c3);
g.drawLine(20, 150, 400, 40);
g.drawLine(5, 290, 80, 19);
g.setColor(Color.red);
g.drawOval(10, 10, 50, 50);
g.fillOval(70, 90, 140, 100);
g.setColor(Color.blue);
g.drawOval(190, 10, 90, 30);
g.drawRect(10, 10, 60, 50);
g.setColor(Color.cyan);
g.fillRect(100, 10, 60, 50);
g.drawRoundRect(190, 10, 60, 50, 15, 15);
}
}
```

Working with Fonts

Fonts are encapsulated by the Font class. To obtainthe fonts which are available in the system, we can use getAvailableFoontFamilyNames()method defined by GraphicsEnvironmentclass.

```
/ Display Fonts
/*
<applet code="ShowFonts" width=550 height=60>
</applet>
*/
import java.applet.*;
import java.awt.*;
public class ShowFonts extends Applet
{
public void paint(Graphics g)
{
String msg = "";
String FontList[];
GraphicsEnvironment ge =
GraphicsEnvironment.getLocalGraphicsEnvironment();
FontList = ge.getAvailableFontFamilyNames();
for(int i = 0; i < FontList.length; i++)
msg += FontList[i] + " ";
g.drawString(msg, 4, 16);
}
}
```

1.3 Event Handling in Applet

Event

An event is an object that describes a state of change in a source.

Event Source

A source is an object that generates an event.

Event Listeners

A listener is an object that is notified when an event occurs.

Handling Mouse Events

To handle mouse events there are two listeners used.

• MouseListener

• MouseMotionListener

The MouseEvents

There are different mouse events which override with other events. They are

• MouseEntered

• MouseExited

• MouseClicked

• MousePressed

• MouseReleased

• MouseDragged

• MouseMoved

The following program demonstrates the mouse event handlers

```
// Demonstrate the mouse event handlers.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEvents" width=300 height=100>
</applet>
*/
public class MouseEvents extends Applet implements MouseListener,
MouseMotionListener
{
String msg = "";
int mouseX = 0, mouseY = 0;
public void init()
{
addMouseListener(this);
```

```
addMouseMotionListener(this);
}
public void mouseClicked(MouseEvent me)
{
mouseX = 0;
mouseY = 10;
msg = "Mouse clicked.";
repaint();
}
public void mouseEntered(MouseEvent me)
{
mouseX = 0;
mouseY = 10;
msg = "Mouse entered.";
repaint();
}
public void mouseExited(MouseEvent me)
{
mouseX = 0;
mouseY = 10;
msg = "Mouse exited.";
repaint();
}
public void mousePressed(MouseEvent me)
{
mouseX = me.getX();
mouseY = me.getY();
msg = "Down";
repaint();
}
public void mouseReleased(MouseEvent me)
```

```
{
mouseX = me.getX();
mouseY = me.getY();
msg = "Up";
repaint();
}
public void mouseDragged(MouseEvent me)
{
mouseX = me.getX();
mouseY = me.getY();
msg = "*";
showStatus("Dragging mouse at " + mouseX + ", "+ mouseY);
repaint();
}
public void mouseMoved(MouseEvent me)
{
showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
}
public void paint(Graphics g)
{
g.drawString(msg, mouseX, mouseY);
}
}
```

1.4 Controls in Applet

The AWT supports the following types of controls:

- Labels
- Push Buttons
- Check Boxes
- Choice Lists
- Lists
- Scroll Bars

• Text Editing

Labels

A label is an object type of Label, and it containsa string which it displays. Labels are passive controls that do not support any interaction with the user. The string in the default label is left-justified. The alignment can be modified by using Label. LEFT, Label.RIGHT and Label.CENTER.

The following example creates three labels and addsthem to an applet.

```
// Demonstrate Labels
import java.awt.*;
import java.applet.*;
/*
<applet code="LabelDemo" width=300 height=200>
</applet>
*/
public class LabelDemo extends Applet
{
public void init()
{
Label one = new Label("One");
Label two = new Label("Two");
Label three = new Label("Three");
add(one);
add(two);
add(three);
}
}
```

Buttons (Implements ActionListener)

A bush button is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type Button. The label of a button is used to determine which button has been pressed. The label is obtained by calling the getActionCommand() method on the ActionEvent objectpassed to actionPerformed().

Sample program to demonstrate buttons

```
// Demonstrate Buttons
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ButtonDemo" width=250 height=150>
</applet>
*/
public class ButtonDemo extends Applet implements ActionListener
{
String msg = "";
Button yes, no, maybe;
public void init()
{
yes = new Button("Yes");
no = new Button("No");
maybe = new Button("Undecided");
add(yes);
add(no);
add(maybe);
yes.addActionListener(this);
no.addActionListener(this);
maybe.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
String str = ae.getActionCommand();
if(str.equals("Yes"))
{
msg = "You pressed Yes.";
```

```
}
else if(str.equals("No"))
{
msg = "You pressed No.";
}
else
{
msg = "You pressed Undecided.";
}
repaint();
}
public void paint(Graphics g)
{
g.drawString(msg, 6, 100);
}
}
```

Check Boxes (Implements ItemListener)

A check box is an control that is used to turn an option on or off. To retrieve the current state of a box, call getState(). To set this state, call setState(). You can obtain the current checkbox label by calling getLabel() and set the label by calling setLabel(). ItemListener interface defines the itemStateChanged method. An ItemEvent object is supplied as the argument to this method.

Demonstration of check boxes

```
// Demonstrate check boxes.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CheckboxDemo" width=250 height=200>
</applet>
*/
```

```
public class CheckboxDemo extends Applet implementsItemListener
{
String msg = "";
Checkbox Win98, winNT, solaris, mac;
public void init()
{
Win98 = new Checkbox("Windows 98/XP", null, true);
winNT = new Checkbox("Windows NT/2000");
solaris = new Checkbox("Solaris");
mac = new Checkbox("MacOS");
add(Win98);
add(winNT);
add(solaris);
add(mac);
Win98.addItemListener(this);
winNT.addItemListener(this);
solaris.addItemListener(this);
mac.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie)
{
repaint();
}
public void paint(Graphics g)
{
msg = "Current state: ";
g.drawString(msg, 6, 80);
msg = " Windows 98/XP: " + Win98.getState();
g.drawString(msg, 6, 100);
msg = " Windows NT/2000: " + winNT.getState();
g.drawString(msg, 6, 120);
```

```
msg = " Solaris: " + solaris.getState();
g.drawString(msg, 6, 140);
msg = " MacOS: " + mac.getState();
g.drawString(msg, 6, 160);
}
}
```

Check Box Group (Implements ItemListener)

Check box group allows the check boxes can be selected at any one time. To determine which check box is currently selected by calling getSelectedCheckbox(). To set a check box by calling setSelectedCheckbox().

Here is a program that uses check boxes that are part of group.

```
// Demonstrate check box group.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CBGroup" width=250 height=200>
</applet>
*/
public class CBGroup extends Applet implements ItemListener
{
String msg = "";
Checkbox Win98, winNT, solaris, mac;
CheckboxGroup cbg;
public void init()
{
cbg = new CheckboxGroup();
Win98 = new Checkbox("Windows 98/XP", cbg, true);
winNT = new Checkbox("Windows NT/2000", cbg, false);
solaris = new Checkbox("Solaris", cbg, false);
mac = new Checkbox("MacOS", cbg, false);
```

```
add(Win98);
add(winNT);
add(solaris);
add(mac);
Win98.addItemListener(this);
winNT.addItemListener(this);
solaris.addItemListener(this);
mac.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie)
{
repaint();
}
public void paint(Graphics g)
{
msg = "Current selection: ";
msg += cbg.getSelectedCheckbox().getLabel();
g.drawString(msg, 6, 100);
}
}
```

Text Field (Implements ActionListener)

The text field implements a single line text entryarea, usually called an edit control. To obtain the string currently contained in the text field, call getText() and to set the text, call setText().

Here is an example that creates the classic username and password screen

```
// Demonstrate text field.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="TextFieldDemo" width=380 height=150>
```

```
</applet>
*/
public class TextFieldDemo extends Applet implements ActionListener
{
TextField name, pass;
public void init()
{
Label namep = new Label("Name: ", Label.RIGHT);
Label passp = new Label("Password: ", Label.RIGHT);
name = new TextField(12);
pass = new TextField(8);
pass.setEchoChar('?');
add(namep);
add(name);
add(passp);
add(pass);
name.addActionListener(this);
pass.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
repaint();
}
public void paint(Graphics g)
{
g.drawString("Name: " + name.getText(), 6, 60);
g.drawString("Password: " + pass.getText(), 6,100);
}
}
```

Chap 2. Swing Programming

2.1 Swing Basics

Java Foundation Classes (JFC) consists of number of class libraries that can be used to construct GUI programs. AWT and swing are the two libraries in JFC.
Swing is a set of classes that provides more powerful and flexible components which can have look and feel concept for platform-independent which again known as light weight elements.
Swing related classes are contained in javax.swing and its sub packages.
Fundamental to swing is the JApplet class, which extends Applet. When adding a component to JApplet as like Applet we can't use add(). Instead of add(), JApplet using add() for the content pane.
Swing Classes
There are number of classes in swing packages. Some of them are,

Class	Description
ImageIcon	Encapsulates an icon
JApplet	The swing version of Applet
JButton	The swing push button class
JCheckBox	The swing check box class
JComboBox	The Swing combo box
JLabel	The swing version of label
JRadioButton	The swing version of a radio button
JScrollPane	Encapsulates a scrollable window
JTabbedPane	Encapsulates a tabbed window
JTable	Encapsulates a table-based control
JTextField	The wing version of a text field
JTree	Encapsulates a tree-based control

How to run Swing programs

1. Compose and compile the swing program with the extension .java
2. Execute the swing program as like the normal java program execution.

2.2 Controls in Swing

Icons and Labels

In swing icons are encapsulated by the ImageIcon class, which paints an icon from and image. Swing labels are instances of the JLabel class. It can display text and / or an icon.

The following example illustrates how to create anddisplay a label containing both an icon and a string.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JLabelDemo" width=250 height=150>
</applet>
*/
public class JLabelDemo extends JApplet
{
public void init()
{
Container contentPane = getContentPane();
ImageIcon ii = new ImageIcon("france.gif");
JLabel jl = new JLabel("France", ii, JLabel.CENTER);
contentPane.add(jl);
}
}
```

Text Fields

JTextFiled allows you to edit one line of text. Itcan be specified with the number of columns in the text field. The following exampleillustrates how to create a text field.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JTextFieldDemo" width=300 height=50>
```

```
</applet>
*/
public class JTextFieldDemo extends JApplet
{
JTextField jtf;
public void init()
{
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
jtf = new JTextField(15);
contentPane.add(jtf);
}
}
```

Buttons

The JButton class provides the functionality of a push button. JButton allows an icon a string, or both to be associated with the push button. The following example displays four push buttons and a text field.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JButtonDemo" width=250 height=300>
</applet>
*/
public class JButtonDemo extends JApplet implements ActionListener
{
JTextField jtf;
public void init()
{
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
```

```
ImageIcon france = new ImageIcon("france.gif");
JButton jb = new JButton(france);
jb.setActionCommand("France");
jb.addActionListener(this);
contentPane.add(jb);
ImageIcon germany = new ImageIcon("germany.gif");
jb = new JButton(germany);
jb.setActionCommand("Germany");
jb.addActionListener(this);
contentPane.add(jb);
ImageIcon italy = new ImageIcon("italy.gif");
jb = new JButton(italy);
jb.setActionCommand("Italy");
jb.addActionListener(this);
contentPane.add(jb);
ImageIcon japan = new ImageIcon("japan.gif");
jb = new JButton(japan);
jb.setActionCommand("Japan");
jb.addActionListener(this);
contentPane.add(jb);
jtf = new JTextField(15);
contentPane.add(jtf);
}
public void actionPerformed(ActionEvent ae)
{
jtf.setText(ae.getActionCommand());
}
}
```

Check Boxes

The JCheckBox class, which provides the functionality of check box. When a check box is selected or deselected, an item event is generated. This is handled by

itemStateChanged().

The following example illustrates how to create an applet that displays four check boxes and a text field.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JCheckBoxDemo" width=400 height=50>
</applet>
*/
public class JCheckBoxDemo extends JApplet implements ItemListener
{
JTextField jtf;
public void init()
{
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
ImageIcon normal = new ImageIcon("normal.gif");
ImageIcon rollover = new ImageIcon("rollover.gif");
ImageIcon selected = new ImageIcon("selected.gif");
JCheckBox cb = new JCheckBox("C", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("C++", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("Java", normal);
```

```
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("Perl", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
jtf = new JTextField(15);
contentPane.add(jtf);
}
public void itemStateChanged(ItemEvent ie)
{
JCheckBox cb = (JCheckBox)ie.getItem();
jtf.setText(cb.getText());
}
}
```

Radio Buttons

Radio buttons are supported by JRadioButton class. Radio buttons must be configured into a group. The ButtonGroup class is instantiated to create a button group. Radio button presses generate action events that are handled by actionPerformed(). The getActionCommand() method gets the text associated with a radio button.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JRadioButtonDemo" width=300 height=50>
</applet>
*/
public class JRadioButtonDemo extends JApplet implements ActionListener
```

```
{
JTextField tf;
public void init()
{
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
JRadioButton b1 = new JRadioButton("A");
b1.addActionListener(this);
contentPane.add(b1);
JRadioButton b2 = new JRadioButton("B");
b2.addActionListener(this);
contentPane.add(b2);
JRadioButton b3 = new JRadioButton("C");
b3.addActionListener(this);
contentPane.add(b3);
ButtonGroup bg = new ButtonGroup();
bg.add(b1);
bg.add(b2);
bg.add(b3);
tf = new JTextField(5);
contentPane.add(tf);
}
public void actionPerformed(ActionEvent ae)
{
tf.setText(ae.getActionCommand());
}
}
```

Combo Boxes

Swing provides a combo box (a combination of a textfield and a drop down list) through the JComboBox class. A combo box normally displays one entry. It can also

display a drop down list that allows a user to select a different entry. Items are added to the list of choices via the addItem() method.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JComboBoxDemo" width=300 height=100>
</applet>
*/
public class JComboBoxDemo extends JApplet implements ItemListener
{
JLabel jl;
ImageIcon france, germany, italy, japan;
public void init()
{
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
JComboBox jc = new JComboBox();
jc.addItem("France");
jc.addItem("Germany");
jc.addItem("Italy");
jc.addItem("Japan");
jc.addItemListener(this);
contentPane.add(jc);
jl = new JLabel(new ImageIcon("france.gif"));
contentPane.add(jl);
}
public void itemStateChanged(ItemEvent ie)
{
String s = (String)ie.getItem();
jl.setIcon(new ImageIcon(s + ".gif"));
```

```
}
}
```

Tabbed Panes

A tabbed pane is a componenet that appears as a group of folders in a file or cabinet. Each folder has a title. When a user selects a folder, its contents become visible. Only one of the folders may be selected ata time. Tabbed panes are encapsulated by the JTabbedPane class. To add to the tab to the pane call, addTab().

```
import javax.swing.*;
/*
<applet code="JTabbedPaneDemo" width=400 height=100>
</applet>
*/
public class JTabbedPaneDemo extends JApplet
{
public void init()
{
JTabbedPane jtp = new JTabbedPane();
jtp.addTab("Cities", new CitiesPanel());
jtp.addTab("Colors", new ColorsPanel());
jtp.addTab("Flavors", new FlavorsPanel());
getContentPane().add(jtp);
}
}
class CitiesPanel extends JPanel
{
public CitiesPanel()
{
JButton b1 = new JButton("New York");
add(b1);
JButton b2 = new JButton("London");
add(b2);
```

```
JButton b3 = new JButton("Hong Kong");
add(b3);
JButton b4 = new JButton("Tokyo");
add(b4);
}
}
class ColorsPanel extends JPanel
{
public ColorsPanel()
{
JCheckBox cb1 = new JCheckBox("Red");
add(cb1);
JCheckBox cb2 = new JCheckBox("Green");
add(cb2);
JCheckBox cb3 = new JCheckBox("Blue");
add(cb3);
}
}
class FlavorsPanel extends JPanel
{
public FlavorsPanel()
{
JComboBox jcb = new JComboBox();
jcb.addItem("Vanilla");
jcb.addItem("Chocolate");
jcb.addItem("Strawberry");
add(jcb);
}
}
```

Chap3. Java Beans

3.1 Bean Basics

A Java Bean is a software component that has been designed to be reusable in a variety of different environments. A bean obtains all the benefits of Java's "write-once, run-anywhere" paradigm.

Application Builder Tools

Application builder tool is a utility that enables you to configure a set of beans connect them together to produce a working application. Bean builder tools have the following capabilities.

• A palette is provided that lists all of the available beans.

• A worksheet is displayed that allows the designer to lay out beans in a GUI.

• Special editors and customizers allow a bean to be configured.

• Commands allow a designer to inquire about the state and behavior of a bean.

• Events generated by one component are mapped to method invocations on other components. This capability is known as interconnection of beans.

Tools

There are two tools available to develop beans

1. **Bean Developer Kit (BDK)**
2. **Bean Builder**

The BDK is for use with versions of Java 2 prior to1.4. For Java 2m V1.4 or later, you must use the Bean Builder Tool which is the beta and alpha versions available from Sun Java.

3.2 Using the Bean Developer Kit (BDK)

Here are the steps that you must follow to create a bean in BDK

1. Create a directory for the new bean
2. Create the java source file
3. Compile the source file
4. Create a Manifest file
5. Generate JAR file
6. Start the BDK
7. Test you bean

Steps in detail

1. Creating a directory for the new bean

You need to make a directory for the bean. Create a directory and change in to that directory called c:\bdk\demo\sunw\demo\colors

2. Create the source file

You can create a sample bean called Colors.java under that directory.

```
// A simple Bean.
package sunw.demo.colors;
import java.awt.*;
import java.awt.event.*;
public class Colors extends Canvas
{
transient private Color color;
private boolean rectangular;
public Colors()
{
addMouseListener(new MouseAdapter()
{
public void mousePressed(MouseEvent me)
{
change();
}
});
rectangular = false;
setSize(200, 100);
change();
}
public boolean getRectangular()
{
return rectangular;
}
public void setRectangular(boolean flag)
```

```
{
this.rectangular = flag;
repaint();
}
public void change()
{
color = randomColor();
repaint();
}
private Color randomColor()
{
int r = (int)(255*Math.random());
int g = (int)(255*Math.random());
int b = (int)(255*Math.random());
return new Color(r, g, b);
}
public void paint(Graphics g)
{
Dimension d = getSize();
int h = d.height;
int w = d.width;
g.setColor(color);
if(rectangular) {
g.fillRect(0, 0, w-1, h-1);
}
else
{
g.fillOval(0, 0, w-1, h-1);
}
}
}
```

3. Compile the source code for the new bean

Compile the source code to create a class file. (javac Colors,java)

4. Create a Manifest file

A developer must provide a manifest file to indicate which of the components in a JAR file are Java Beans. A manifest file may reference several .classfiles. If a .class file is a Java Bean, its entry in a JAR file must be immediately followed by the line "Java-Bean: True".

Under the directory c:\bdk\democreate a file called Colors.mft with the following

Name: sunw/demo/colors/Colors.class

Java-Bean: True

5. Generate JAR file

Beans are packaged into JAR (Java Archive) files. Under the directory c:\bdk\jars, use this following command to create a colors.jarfile .

jar cfm ..\jars\colors.jar colors.mft sunw\demo\colors*.class

6. Start the BDK

Change the directory c:\bdk\beanboxand type run.

7. Test your bean

In the bean box you can see three windows, titled ToolBox, BeanBox and Properties. Use the Properties window to changethe rectangularproperty from falseto true. Its shape immediately changes.

3.3 Using the Bean Builder

As explained before BDK is not compatible with Java 2, version 1.4. Instead , 1.4 user will need to use the new Bean Builder tool forBean development. At this time Bean Builder is available only as beta and alpha release, and its final form and feature set are subject to change.

Bean Builder is similar to Bean Box offered by the BDK, except that is more sophisticated and powerful. Its operation is similar to Bean Box except that is easier to use.

You can run the bean builder using the following web site

http://javadesktop.org/jdnc/bean-builder/0_6/beanbuilder.jnlp

Bean Builder Windows

Bean Builder provides three windows.

1. **Top or main window**

Used to hold the current palette set

2. **Property Inspector window**

Used to examine and set the properties each bean component.

3. **Design window**

Design window is the window in which you will assemble various components into an application.

chap4. Java Networking

4.1 Networking Basics

Networking classes in Java are defined in the java.netpackage. These networking classes encapsulate the socket paradigm pioneered in the Berkley Sockets.

Socket

A network socket can be used to implement reliable connections between hosts and Internet.

Client / Server

A server is anything that has some resource that can be shared. A client is simply any other entity that wants to gain access to a particular sever.

Reserved Sockets

TCP/IP reserves the lower 1024 ports for specific protocols. Port number 21 is for FTP, 23 is for Telnet, 25 is for e-mail, 79 is for finger, 80 is for HTTP and 119 is for net news.

Proxy Servers

A proxy server speaks the client side of a protocol to another server. A proxy server has the additional ability to filer certain requests or caches the results of those requests for future use.

Internet Addressing

Every computer on the internet has an address. An internet address is a number that uniquely identifies each computer on the net.

Domain Naming Service (DNS)

Domain naming service is used to refer the four numbers of an IP address described in network hierarchy as domain name of the machine's name.

4.2 Java and Net

InetAddress

The InetAddress class is used to encapsulate both the numerical IP address and domain name. The getLocalHost()method return the InetAddress of the local host. The getByName()method returns an InetAddress for a host name passed to it. The getAllByName()method returns InetAddresses if it is more than one. If these methods are unable to resolve a hostname, they throw an UnKnownHostException.

The following example prints the address and names of the local machine.

```
// Demonstrate InetAddress.
import java.net.*;
class InetAddressTest
{
public static void main(String args[]) throws UnknownHostException
{
InetAddress Address = InetAddress.getLocalHost();
System.out.println(Address);
Address = InetAddress.getByName("yahoo.com");
System.out.println(Address);
InetAddress SW[] = InetAddress.getAllByName("www.nba.com");
for (int i=0; i<SW.length; i++)
System.out.println(SW[i]);
```

Whois

Whois port allows to open a connections on the InterNIC server, sends the domain through the socket and prints the data that is returned. InterNIC will try to look up the domain as internet registered domain name, then send back the IP address and contact information for that site.

```
//Demonstrate Sockets.

import java.net.*;

import java.io.*;

class Whois

{

public static void main(String args[]) throws Exception

{

int c;

Socket s = new Socket("internic.net", 43);

InputStream in = s.getInputStream();

OutputStream out = s.getOutputStream();

String str = “nba.com”;

byte buf[] = str.getBytes();

out.write(buf);

while ((c = in.read()) != -1) {

System.out.print((char) c);

}

s.close();

}

}
```

URL

The Uniform Resource Locator provides intelligible form to uniquely identify or address information on the Internet. The following example we create URL to google's downloaded page and then examine its properties.

```
// Demonstrate URL.
import java.net.*;
class URLDemo
{
public static void main(String args[]) throws MalformedURLException
{
URL hp = new URL("http://www.osborne/downloads");
System.out.println("Protocol: " + hp.getProtocol());
System.out.println("Port: " + hp.getPort());
System.out.println("Host: " + hp.getHost());
System.out.println("File: " + hp.getFile());
System.out.println("Ext:" + hp.toExternalForm());
}
}
```

URLConnection

URLConnection is a general purpose class for accessing the attributes of a remote resource. In the following example, we create a URLConnection using the openConnection()method of a URL object and then use it to examine the properties of

document or content.

```
// Demonstrate URLConnection.

import java.net.*;

import java.io.*;

import java.util.Date;

class UCDemo

{

public static void main(String args[]) throws Exception

{

int c;

URL hp = new URL("http://www.internic.net");

URLConnection hpCon = hp.openConnection();

long d = hpCon.getDate();

if(d==0)

System.out.println("No date information.");

else

System.out.println("Date: " + new Date(d));

System.out.println("Content-Type: " + hpCon.getContentType());

d = hpCon.getExpiration();

if(d==0)

System.out.println("No expiration information.");
```

```
else
System.out.println("Expires: " + new Date(d));
d = hpCon.getLastModified();
if(d==0)
System.out.println("No last-modified information.");
else
System.out.println("Last-Modified: " + new Date(d));
int len = hpCon.getContentLength();
if(len == -1)
System.out.println("Content length unavailable.");
Else
System.out.println("Content-Length: " + len);
if(len != 0) {
System.out.println("=== Content ===");
InputStream input = hpCon.getInputStream();
int i = len;
while (((c = input.read()) != -1)) { // && (--i > 0)) {
System.out.print((char) c);
}
input.close();
} else {
```

```
System.out.println("No content available.");

}

}

}
```

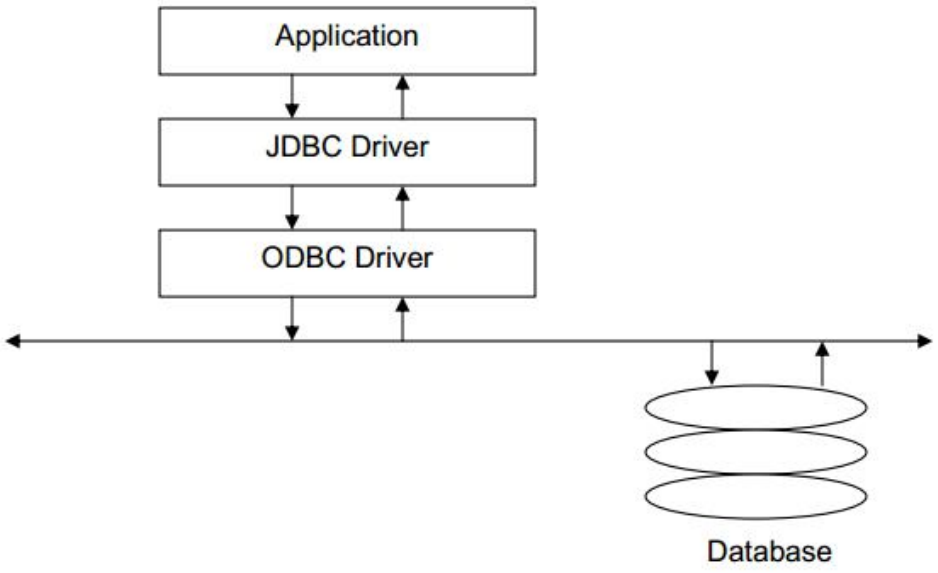
Chap 5. Java Database Connectivity

5.1 JDBC Basics

Java Database Connectivity is used to connect the database applications with Java programs. JDBC is designed to be platform-independent. JDBC Application Programming Interface defines how a program written in Java can communicate and interact with the database.

JDBC-ODBC Bridge

The JDBC-ODBC bridge driver is the only driver supplied by JavaSoft. JDBC driver only can talk with ODBC. Once JDBC passes the request off to the ODBC driver, it the responsibility of the ODBC driver to communicate with the database.



Driver Manager

JDBC provides a driver manager to load the driver using the Class.forName() method. The driver manager for Java JDBC-ODBC is sun.jdbc.odbc.JdbcOdbcDriver.

Connection

A connection object represents a connection with the database.

Statement

A statement object is used to send SQL statements to a database.

ResultSet

A Result Set is an object that contains the result of execution of a SQL query.

5.2 JDBC Programming

Steps in JDBC Programming

1. Find the JDBC Driver

Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);

2. Configure the database

Create a ODBC for your database application by providing a

DataSourceName.

3. Test the configuration by establishing the connection

Connection c=DriverManager. getCOnnection(jdbc:odbc:DSN,user,pws);

4. Generate SQL query and ResultSet.

5. Close Connection and close statement.

Sample program which connects Microsoft Access Database

```
import java.sql.*;
class JdbcDemo
{
public static void main(String args[]))
{
try
{
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
```

```
Connection c=Drivermanager.getConnection("jdbc:odbc:people,"","");
Statement s=c.createStatement();
ResultSet rs=s.executeQuery("select name,address from people");

while(rs.next())
{
System.out.println(rs.getString("name")+","+rs.getString("address"));
}
s.close();
}
catch(Exception e)
{
 System.out.println("Exception"+e);
}
c.close();
}
}
```

To Create a Table

```
import java.sql.*;
class createtable
{
public static void main(String args[])
{
try
{
```

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn=DriverManager.getConnection("jdbc:odbc:shan","sub","ssi");
Statement stmt=conn.createStatement();
stmt.execute("create table sha(Eno varchar(20),ename varchar(20))");
System.out.println("Table Created");
}catch(SQLException e)
{
 System.out.println(e);
}
catch(ClassNotFoundException e)
{  System.out.println(e);
}
}
}
```

Simple Insert into Table

```
import java.sql.*;
class inserttable
{
public static void main(String args[])
{
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn=DriverManager.getConnection("jdbc:odbc:shiv","sub","ssi");
Statement stmt=conn.createStatement();
```

```
stmt.execute("insert into sha values(101,'sachin')");
System.out.println("Row inserted");
}catch(SQLException e)
{
 System.out.println(e);
}
catch(ClassNotFoundException e)
{  System.out.println(e);
}
}
}
```

Using Prepared Statement to Insert into Table

```
import java.sql.*;
class psinsert
{
public static void main(String args[]) throws IOException
{
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn=DriverManager.getConnection("jdbc:odbc:shan","sub","ssi");
DataInputStream d1=new DataInputStream(System.in);
System.out.print("Enter Emp No : ");
String u=d1.readLine();
System.out.print("Enter Employee Name : ");
```

```
String p=d1.readLine();
PreparedStatement pstmt=conn.prepareStatement("Insert into sha values(?,?)");
pstmt.setString(1,u);
pstmt.setString(2,p);
pstmt.execute();
System.out.println("One Row Inserted");
}catch(SQLException e)
{System.out.println(e);}
catch(ClassNotFoundException e)
{System.out.println(e);}
}
}
```

JDBC Application Using GUI

```
import java.awt.*;
import javax.swing.*;
import java.sql.*;
import java.io.*;
public class Students extends JFrame
{
 public Students()
 {
 super("Student Information");
  try
  {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
 Connection con
=DriverManager.getConnection("jdbc:odbc:studentdsn");
 Statement st=con.createStatement();
 String s1=JOptionPane.showInputDialog("Enter a");
 int a=Integer.parseInt(s1);
 String s2=JOptionPane.showInputDialog("Enter b");
 String s3=JOptionPane.showInputDialog("Enter c");
 ResultSet rs=st.executeQuery("Select * from student");
 ResultSetMetaData meta=rs.getMetaData();
int x=0;
while(rs.next())
 {
  x=x+1;
  }
rs.absolute(x);
 rs.updateInt(1,a);
 rs.updateString(2,s2);
 rs.updateString(3,s3);
 rs.updateRow();
 StringBuffer s=new StringBuffer();
 int cols=meta.getColumnCount();
 for(int i=1;i<=cols;i++)
 s.append(meta.getColumnName(i)+"\t");
 s.append("\n");
 rs.beforeFirst();
```

```
   while(rs.next())
{
    for(int i=1;i<=cols;i++)
    s.append(rs.getObject(i)+"\t");
    s.append("\n");
   }
   JTextArea area=new JTextArea(s.toString());
   getContentPane().add(new JScrollPane(area));
   setSize(300,300);
   setVisible(true);
   st.close();
   con.close();
  }
  catch(Exception e)
  {
JOptionPane.showMessageDialog(null,e, "Error",JOptionPane.ERROR_MESSAGE);
   System.exit(1);
  }
 }
 public static void main(String args[])
 {
  Students window=new Students();
  window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
}
```

6. Remote Method Invocation

Remote Method Invocation (RMI) allows a java objectexecutes on one machine to invoke a method of a java object that executes on another machine.

A Simple Client/Sever Application using RMI

Step 1: Enter and compile source code

a) AddServerIntf.java

```
import java.rmi.*;
public interface AddServerIntf extends Remote
{
double add(double d1, double d2) throws RemoteException;
}
```

b) AddServerImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
public class AddServerImpl extends UnicastRemoteObject implements AddServerIntf
{
public AddServerImpl() throws RemoteException { }
public double add(double d1, double d2) throws RemoteException
{
return d1 + d2;
}
}
```

c) AddServer.java

```
import java.net.*;
import java.rmi.*;
```

```
public class AddServer
{
public static void main(String args[])
{
try
{
AddServerImpl addServerImpl = new AddServerImpl();
Naming.rebind("AddServer", addServerImpl);
}
catch(Exception e)
{
System.out.println("Exception: " + e);
}
} }
```

d) AddClient.java

```
import java.rmi.*;
public class AddClient
{
public static void main(String args[])
{
try
{
String addServerURL = "rmi://" + args[0] + "/AddServer";
AddServerIntf addServerIntf =
(AddServerIntf)Naming.lookup(addServerURL);
```

```
System.out.println("The first number is: " + args[1]);
double d1 = Double.valueOf(args[1]).doubleValue();
System.out.println("The second number is: " +args[2]);
double d2 = Double.valueOf(args[2]).doubleValue();
System.out.println("The sum is: " + addServerIntf.add(d1, d2));
}
catch(Exception e)
{
System.out.println("Exception: " + e);
}
}
}
```

Step 2: Generate Stubs and Skeletons

Stub

A stub is a java object that resides on the client machine. Its function to present the same interfaces as the remote server.

Skeleton

A skeleton is a java object that resides on a server machine. It works with the other parts of RMI system to receive the requests and invoke appropriate code on the server.

To generate Stubs and Skeletons

To generate stubs and skeletons use the tool RMI compiler, this is invoked from the command line as shown as shown here:

rmic AddServerImpl

This command generates two new files: AddServerImpl_Skel.class

AddServerImpl_Stub.class

Step 3: Install files on the client and server machines

Under Client Machine copy the following files:

AddClient.class

AddServerImpl_Stub.class

AddServerIntf.class

Under Server Machine copy the following files:

AddServerIntf.class

AddServerImpl.class

AddServerImpl_Skel.class

AddServerImpl_Stub.class

AddServer.class

Step 4: Start the RMI Registry on the Server Machine

On the server machine, type the following in the command line

start rmiregistry

Step 5: Start the Server

The server code is started from the command line, as shown as here:

java AddServer

Step 6: Start the Client

The AddClient software requires three argunments: the name or IP address of the server and the two numbers that are to be summed together. From the command line use any one of the following command,

java AddClient server1 8 9

java AddClient 192.168.0.1 8 9