UNIT 2 INHERITANCE AND POLYMORPHISM

Structure **Page Nos.**

2.0 INTRODUCTION

In Object Oriented Programming, one of the major feature is reusability. You are already introduced to the reusability concept in Unit 2 of Block1 of this course. Reusability in Java is due to the inheritance. Inheritance describes the ability of one class or object to possess characteristics and functionality of another class or object in the hierarchy. Inheritance can be supported as "run-time" or "compile-time" or both. Here we are discussing inheritance with the Java programming language, and in our discussion we will generally refer to compile-time inheritance.

In this unit we will discuss importance of inheritance in programming, concept of superclass and subclass, and access controls in Java programming language. You will see through example programs how methods are overridden, and how methods of a super class are accessed by subclass. You will also see how multilevel inheritance is implemented, use of abstract classes, and demonstration of polymorphism with the help of example program.

2.1 OBJECTIVES

After going through this unit you will be able to:

- explain the need of inheritance;
- write programs to demonstrate the concept of super class and sub class;
- describe access control in Java;
- explain method overriding and its use in programming;
- incorporate concept of multilevel inheritance in programming, and
- define abstract classes and show use of polymorphism in problem solving.

2.2 INHERITANCE BASICS

Inheritance is a language property specific to the object-oriented paradigm. Inheritance is used for a unique form of code-sharing by allowing you to take the implementation of any given class and build a new class based on that implementation. Let us say class B, starts by inheriting all of the data and operations defined in the class A. This new subclass can extend the behaviour by adding additional data and new methods to operate on it. Basically during programming inheritance is used for extending the existing property of a class. In other words it can be said that inheritance is "from generalization- to-specialization". In this by using general class, class with specific properties can be defined.

Now let us take the example of Employee class, which is declared as:

```
class BankAccount
{
data members
member functions
}
```

data members and member functions of BankAccount class are used to display characteristics of Withdraw, Deposit, getBalance of objects of a BankAccount class.

Now suppose you want to define a SavingAccount class. In the SavingAccount class definitely you will have basic characteristics of a Bankaccount class mentioned above. In your SavingAccount class you can use the properties of BankAccount class, without any modification in them. This use of properties of BankAccount class in Saving Account class is called inheriting property of BankAccount class into SavingAccount class.

To inherit a class into another class **extends** keyword is used. For example, SavingAccount class will inherit BankAccount class as given below.

```
class SavingAccount extends BankAccount
{
data members
ember functions
}
```

In this example, SavingAccount declares that it inherits or "extends" BankAccount.

Now let us see what is superclass and subclass.

Superclass and Subclass

The superclass of a class A is the class from which class A is derived. In programming languages like C++, allow deriving a class from multiple classes at a time. When a class inherits from multiple super classes, the concepts is known as multiple inheritance. Java doesn't support multiple inheritance. If there is a need to implement multiple inheritance. It is realized by using interfaces. You will study about Interfaces in Unit 4 of this Block.

A class derived from the superclass is called the subclass. Sometime-superclass is also called parent class or base class and subclass is called as child class or derived class. The subclass can reuse the data member and methods of the superclass that were already implemented and it can also extend or replace the behaviour in the superclass by overriding methods. Subclass can have its own data members and member functions.

You can see in this example program, the Employee class is used for tracking the hours an employ worked for along with the hourly wages, and the "attitude" which gives you a rough measure of their activeness or for what percentage of time they are actually productive.

```
public class Employee
{
protected double attitude;
protected int numHoursPerWeek, wagePerHour;
public Employee(int wage, int hours, double att) // constructor
{
```

```
wagePerHour = wage;
numHoursPerWeek = hours;
attitude = att;
}
public double getProductivity()
{
return numHoursPerWeek*attitude;
}
public double getTeamProductivity()
{
return getProductivity();
}
public int WeekSalary()
{
return wagePerHour*numHoursPerWeek;
}
}
```

If you look closely you will observe that Employee class possesses the very basic characteristics of an employee. So think quickly about different type of employees! Of course you can think about employees with special characteristics, for example, Manager Engineer, Machine-man etc. You are right Subclass of Employee, will have properties of Employee class as well as some more properties.

For example, if you take a class Manager (Subclass of Employee class) class. The Manager is a more specialized kind of Employee. An important point to note is that Manager represents ***a*** relationship with Employee. A Manager is a particular type of employee, and it has all the properties of Employee class plus some more property specific to it. Manager overrides the team productivity method to add the work done by the employees working under him/her and adds some new methods of its own dealing with other properties, which reflect characteristics of typical Managers. For example, annoying habits, taking employees under her/him, preparing report for employees, etc.

```
public class Manager extends Employee
{
// subclass of Employee
public Manager(int wage, int hours, double att, Employee underling)
{
super(wage, hours, att); // chain to our superclass constructor
 }
public double getTeamProductivity()
{
// Implementation here
}
public int askSalary()
{
  // Implementation here
}
public void addUnderling(Employee anUnderling)
{
// Implementation here
}
public void removeUnderling(Employee anUnderling)
{
// Implementation here
}
```

In the incomplete program above you can see that how inheritance is supporting in cremental development. Basically in the above program (this program is incomplete, you can write code to complete it) an attempt has been made to introduce a new code, without causing bugs in the existing code.

2.3 ACCESS CONTROL

You can see that the terms super, public and protected are used in previous programs. Can you tell what is the role of these terms in programs? Right, public and protected are access controller, used to control the access to members (data members and member functions) of a class, and super is used in implementing inheritance.

Now you can see how access control is used in Java programs.

Controlling Access to Members of a Class

One of the objectives of having access control is that classes can protect their member data and methods from getting accessed by other objects. Why is this important? Well, consider this. You're writing a class that represents a query on a database that contains all kinds of secret information; say student's records or marks obtained by a student in final examination.

In your program you will have certain information and queries contained in the class. Class will have some publicly accessible methods and variables in your query object, and you may have some other queries contained in the class simply for the personal use of the class. These methods support the operation of the class but should not be used by objects of another type. In other words you can say–you've got secret information to protect.

How can you protect it?

Ok in Java, you can use access specifiers to protect both variables and methods of a class when you declare them. The Java language supports four distinct access specifiers for member data and methods: private, protected, public, and if left unspecified, package.

The following chart shows the access level permitted by each specifier.

Specifier	**class**	**subclass**	**package**	**world**
Private	X			
Protected	X	X*	X	
Public	X	X	X	X
Package	X		X	

The first column indicates whether the class itself has access to the members defined by the access specifier. As you can see, a class always has access to its own members.

The second column indicates whether subclasses of the class (regardless of which package they are in) have access to the member.

The third column indicates whether classes in the same package as the class (regardless of their parentage) have access to the member.

The fourth column indicates whether all classes have to the member.

Note that the protected/subclass intersection has an '*'. This particular case has a special association with inheritance implementation. You will see in the next section of this unit how protected specifier is used in inheritance. Package will cover in the next unit of this Block.

☞ Check Your Progress 1

1) What is the advantage of inheritance? How can a class inherit the properties of any other class in Java?

……………………………………………………………………………………

……………………………………………………………………………………

2) Explain the need of access specifiers.

……………………………………………………………………………………

……………………………………………………………………………………

3) When is private specifier used in a program?

……………………………………………………………………………………

……………………………………………………………………………………

Let's look at each access level in more detail.

private

Private is the most restrictive access level. A private member is accessible only to the class in which it is defined. You should use this access to declare members that you are going to use within the class only. This includes variables that contain information if it is accessed by an outsider could put the object in an inconsistent state, or methods, if invoked by an outsider, could jeopardize the state of the object or the program in which it is running. You can see private members like secrets you never tell anybody.

To declare a private member, use the private keyword in its declaration. The following class contains one private member variable and one private method:

```
class First
{
private int MyPrivate; // private data member
private void privateMethod() // private member function
{
System.out.println("Inside privateMethod");
}
}
```

Objects of class First can access or modify the MyPrivate variable and can invoke privateMethod.Objects of other than class First cannot access or modify MyPrivate variable and cannot invoke privateMethod . For example, the Second class defined here:

```
class Second {
void accessMethod() {
First a = new First();
a. MyPrivate = 51;      // illegal
a.privateMethod();      // illegal
}
}
```

cannot access the MyPrivate variable or invoke privateMethod of the object of First.

If you are attempting to access a method to which it does not have access in your program, you will see a compiler error like this:

```
Second.java:12: No method matching privateMethod()
found in class First.
a.privateMethod();        // illegal
1 error
```

One very interesting question can be asked, "whether one object of class First can access the private members of another object of class First". The answer to this question is given by the following example. Suppose the First class contained an instance method that compared the current First object (this) to another object based on their iamprivate variables:

```
class Alpha
{
private int MyPrivate;
boolean isEqualTo (First anotherObject)
{
if (this. MyPrivate == anotherobject. MyPrivate)
return true;
else
return false;
}
}
```

This is perfectly legal. Objects of the same type have access to one another's private members. This is because access restrictions apply at the class or type level (all instances of a class) rather than at the object level.

Now let us discuss protected specifier.

protected

Protected specifiers allows the class itself, subclasses, and all classes in the same package to access the members. You should use the protected access level for those data members or member functions of a class, which you can be accessed by subclasses of that class, but not unrelated classes. You can see protected members as family secrets–you don't mind if the whole family knows, and even a few trusted friends but you wouldn't want any outsiders to know. A member can be declared protected using keyword protected.

```
public class Student
{
protected int age;
public  String name;
protected void protectedMethod()
{
System.out.println("protectedMethod");
}
}
```

You will see the use of protected specifier in programs discussed in next sections of this block.

public

This is the easiest access specifier. Any class, in any package, can access the public members of a class's. Declare public members only if you want to provide access to a

member by every class. In other words you can say if access to a member by outsider cannot produce undesirable results the member may be declared public.
To declare a public member, use the keyword public. For example,

```
public class Account
{
public String name;
protected String Address;
protected int Acc_No;
public void publicMethod()
{
System.out.println("publicMethod");
}
}
class Saving_Account
{
void accessMethod()
{
Account a = new Account();
String MyName;
a.name =  MyName;       // legal
a.publicMethod();       // legal
}
}
```

As you can see from the above code snippet, Saving_Account can legally inspect and modify the name variable in the Account class and can legally invoke publicMethod also.

Member Access and Inheritance

Now we will discuss uses of **super** keyword in Java programming.

There are two uses of super keyword.

1. It is used for calling superclass constructor.
2. It is used to access those members of superclass that are hidden by the member of subclass (How a subclass can hide member of a superclass?).

Can you tell why subclass is called constructor of superclass?

An Object of class is created by call constructor to initialize its data member! Now if you create an object of a subclass you will call a suitable constructor of that subclass to initialize its data members. Can you tell how those data members of the parent class, which subclass is inheriting will be initialized? Therefore, to initialize superclass (parent class) data member, superclass constructor is called in subclass constructor.

To call a superclass constructor write super **(argument-list)** in subclass constructor and this should be the very first statement in the subclass constructor. This argument list includes the arguments needed by superclass constructor. Since the constructor can be overloaded, super () can be called using any form defined by the superclass. In case of constructor overloading in superclass, which of the constructors will be called is decided by the number of parameters or the type of parameter passed in super( ).

Now let us take one example program to show how subclass constructor calls superclass constructor.

```
class  Student
{
```

```
String name;
String address;
int age;
Student( String a, String b, int c)
{
name = a;
address = b;
age = c;
}
void display( )
{
System.out.println("*** Student Information ***");
Sstem.out.println("Name : "+name+"\n"+"Address:"+address+"\n"+"Age:"+age);
}
}
class PG_Student extends Student
{
int age;
int  percentage;
String course;
PG_Student(String a, String b, String c, int d , int e)
{
super(a,b,d);
course = c;
percentage = e;
age = super.age;
}
void display()
{
super.display();
System.out.println("Course:"+course);
}
}
class Test_Student
{
public static void main(String[] args)
{
 Student std1 = new Student("Mr. Amit Kumar" , "B-34/2 Saket J Block",23);
PG_Student pgstd1 = new  PG_Student("Mr.Ramjeet ", "241- Near Fast Lane Road
Raipur" ,"MCA", 23, 80);
std1.display();
pgstd1.display();
}
}
```

Output:

```
*** Student Information ***
Name : Mr. Amit Kumar
Address:B-34/2 Saket J Block
Age:23
*** Student Information ***
Name : Mr.Ramjeet
Address:241- Near Fast Lane Road Raipur
Age:23
Course:MCA
```

In the above program PG_Student class is derived from Student class. PG_Student class constructor has called constructor of Student class. One interesting point to note in this program is that both Student and PG_Student classes have variable named age.

When PG_Student class will inherit class Student, member data **age** of Student class will be hidden by the member data **age** of PG_Student class. To access member data **age** of Student class in PG_Student class, **super.age** is used. Whenever any member of a superclass have the same name of the member of subclass, it has to be accessed by using super keyword prefix to it.

2.4 MULTILEVEL INHERITANCE

Now let us discuss about multilevel. In program given below it is soon that how multilevel inheritance is implemented.

Order of Constructor Calling in Multilevel Inheritance

When the object of a subclass is created the constructor of the subclass is called which in turn calls constructor of its immediate superclass. For example, if we take a case of multilevel inheritance, where class B inherits from class A. and class C inherits from class B. You can see the output of the example program given below, which show the order of constructor calling.

```
//Program
class A
{
A()
{
System.out.println("Constructor of Class A has been called");
}
}
class B extends A
{
B()
{
super();
System.out.println("Constructor of Class B has been called");
}
}
class C extends B
{
C()
{
super();
System.out.println("Constructor of Class C has been called");
}
}
class Constructor_Call
{
public static void main(String[] args)
{
System.out.println("------Welcome to Constructor call Demo------");
C objc = new C();
}
}
```

Output:

```
------Welcome to Constructor call Demo------
Constructor of Class A has been called
Constructor of Class B has been called
Constructor of Class C has been called
```

2.5 METHOD OVERRIDING

You know that a subclass extending the parent class has access to all the non-private data members and methods its parent class. Most of the time the purpose of inheriting properties from the parent class and adding new methods is to extend the behaviour of the parent class. However, sometimes, it is required to modify the behaviour of parent class. To modify the behaviour of the parent class overriding is used.

Some important points that must be taken care while overriding a method:

i. An overriding method (largely) replaces the method it overrides.
ii. Each method in a parent class can be overridden at most once in any one of the subclass.
iii. Overriding methods must have exactly the same argument lists, both in type and in order.
iv. An overriding method must have exactly the same return type as the method it overrides.
v. Overriding is associated with inheritance.

The following example program shows how member function area () of the class Figure is overridden in subclasses Rectangle and Square.

```
class Figure
{
double sidea;
double sideb;
Figure(double a, double b)
{
sidea = a;
sideb = b;
}
Figure(double a)
{
sidea = a;
sideb = a;
}
double area( )
{
System.out.println("Area inside figure is Undefined.");
return 0;
}
}
class Rectangle extends Figure
{
Rectangle( double a , double b)
{
super ( a, b);
}
double area ( )
{
System.out.println("The Area of Rectangle:");
return sidea*sideb;
}
}
class Squre extends Figure
{
Squre( double a )
{
```

```
super (a);
}
double area( )
{
System.out.println("Area of Squre: ");
return sidea*sidea;
}
}
class Area_Overrid
{
public static void main(String[] args)
{
Figure f = new  Figure(20.9, 67.9);
Rectangle r = new  Rectangle( 34.2, 56.3);
Squre s = new Squre( 23.1);
System.out.println("***** Welcome to Override Demo ******");
f.area();
System.out.println(" "+r.area());
System.out.println(" "+s.area());
}
}
```

Output:

```
***** Welcome to Override Demo ******
Area inside figure is Undefined.
The Area of Rectangle:
 1925.46
Area of Squre:
 533.61
```

In most of the object oriented programming languages like C++ and Java, a reference parent class object can be used as reference to the objects of derived classes. In the above program to show overriding feature object of *Figure class* can be used as reference to objects of Rectangle and Square class. Above program with modification(shown in bold) in Area_Override class will be as :

```
class Area_Overrid
{
public static void main(String[] args)
{
Figure f = new  Figure(20.9, 67.9);
Rectangle r = new  Rectangle( 34.2, 56.3);
Squre s = new Squre( 23.1);
System.out.println("***** Welcome to Override Demo ******");
f.area();
f= r;
System.out.println(" "+f.area());
f = s;
System.out.println(" "+f.area());
}
}
```

☞ Check Your Progress 2

1) State True/False for the following statements:

T	F

i. One object can access the private member of the object of the same class. ☐

ii A subclass cannot call the constructor of its super class. ☐

iii A public variable in a package cannot be accessed from other package. ☐

2) Explain the use of super keyword in Java programming.

……………………………………………………………………………………
……………………………………………………………………………………
…………………………………………………………………………………..

3) How is method overriding implemented in Java? Write the advantage of method overriding.

……………………………………………………………………………………
……………………………………………………………………………………
…………………………………………………………………………………..

2.6 ABSTRACT CLASSES

As seen from the previous examples, when we extending an existing class, we have a choice whether to redefine the methods of the superclass. Basically a superclass has common features that are shared by subclasses. In some cases you will find that superclass cannot have any instance (object) and such of classes are called **abstract classes**. Abstract classes usually contain **abstract methods.** Abstract method is a method signature (declaration) without implementation. Basically these abstract methods provide a common interface to different derived classes. Abstract classes are generally used to provide common interface derived classes. You know a superclass is more general than its subclass(es). The superclass contains elements and properties common to all of the subclasses. Often, the superclass will be set up as an *abstract* class, which does not allow objects of its prototype to be created. In this case only objects of the subclass are created. To do this the reserved word *abstract* is included (prefixed) in the class definition.

For example, the class given below is an abstract class.

```
public abstract class Player  // class is abstract
{
private String name;
public Player(String  vname)
{
name=vname;
}
public String getName()        // regular method
{
return (name);
}
public abstract void Play();
//  abstract method: no implementation
}
```

Subclasses *must* provide the method implementation for their particular meaning. If the method statements is one provided by the superclass, it would require overriding in each subclass. In case you forget to override, the applied method statements may be inappropriate.

Now can you think what to do if you have to force that derived classes must redefine the methods of superclass?

The answer is very simple Make those methods abstract.

In case attempts are made to create objects of abstract classes, the compiler doesn't allow and generates an error message. If you are inheriting in a new class from an abstract class and you want to create objects of this new class, you must provide definitions to all the abstract methods in the superclass. If all the abstract methods of super class are not defined in this new class this class also will become abstract.

Is it possible to have an abstract class without abstract method? Yes, you can have. Can you think about the use of such abstract classes? These types of classes are defined in case it doesn't make any sense to have any abstract methods, in the class and yet you want to prevent an instance of that class.

Inheritance represent, "is–a" relationship between a subclass and a superclass. In other words, you can say that every object of a subclass is also a superclass object with some additional properties. Therefore, the possibility of using a subclass object in place of a superclass object is always there. This concept is very helpful in implementing polymorphism.

Now we will discuss polymorphism one of the very important features of object oriented programming, called polymorphism supported by Java programming language.

2.7 POLYMORPHISM

Polymorphism is the capability of a *method* to do different things based on the object through which it is invoked or object it is acting upon. For example method *find _area* will work definitely for Circle object and Triangle object In Java, the type of actual object always determines method calls; object reference type doesn't play any role in it. You have already used two types of polymorphism (overloading and overriding) in the previous unit and in the current unit of this block. Now we will look at the third: *dynamic method binding*. Java uses Dynamic Method Dispatch mechanism to decide at run time which overridden function will be invoked. Dynamic Method Dispatch mechanism is important because it is used to implement runtime polymorphism in Java. Java uses the principle: "a super class object can refer to a subclass object" to resolve calls to overridden methods at run time.

If a superclass has method that is overridden by its subclasses, then the different versions of the overridden methods are invoked or executed with the help of a superclass reference variable.

Assume that three subclasses (Cricket_Player Hockey_Player and Football_Player) that derive from Player abstract class are defined with each subclass having its own Play() method.

```
abstract class Player  // class is abstract
{
private String name;
public Player(String nm)
{
name=nm;
}
public String getName()        // regular method
{
return (name);
}
public abstract void Play();
//  abstract method: no implementation
}
class Cricket_Player extends Player
{
Cricket_Player( String  var)
{
}
public void Play()
{
System.out.println("Play Cricket:"+getName());
}
}
class Hockey_Player extends Player
{
Hockey_Player( String  var)
{
}
public void Play()
{
System.out.println("Play Hockey:"+getName());
}
}
class Football_Player extends Player
{
Football_Player( String  var)
{
}
public void Play()
{
System.out.println("Play Football:"+getName());
}
}
public class PolyDemo
{
public static void main(String[] args)
{
Player ref;                    // set up var for an Playerl
Cricket_Player  aCplayer = new Cricket_Player("Sachin");  // makes specific objects
Hockey_Player aHplayer = new Hockey_Player("Dhanaraj");
Football_Player aFplayer = new Football_Player("Bhutia");
// now reference each as an Animal
ref = aCplayer;
ref.Play();
ref = aHplayer;
ref.Play();
ref = aFplayer;
ref.Play();
}
}
```

Output:

```
Play Cricket:Sachin
Play Hockey:Dhanaraj
Play Football:Bhutia
```

Notice that although each method is invoked through ref, which is a reference to player class (but no player objects exist), the program is able to resolve the correct method related to the subclass object at runtime. This is known as dynamic (or late) method binding.

2.8 FINAL KEYWORD

In Java programming the *final* key word is used for three purposes:

i. Making constants
ii. Preventing method to be overridden
iii. Preventing a class to be inherited

The final keyword (as discussed in Unit 3 of Block1 of this course) is a way of marking a variable as "read-only". Its value is set once and then cannot be changed.

For example, if year is declared as
final int year = 2005;

Variable year will be containing value 2005 and cannot take any other value after words.

The final keyword can also be applied to methods, with similar semantics: i.e. the definition will not change. You cannot override a final method in subclasses, this means the definition is the "final" one. You should define a method final when you are concerned that a subclass may accidentally or deliberately redefine the method (override).

If you want to prevent a class to be inherited, apply the final keyword to an entire class definition. For example, if you want to prevent class Personal to be inherited further, define it as follows:

```
final   class Personal
{
// Data members
//Member functions
}
```

Now if you try to inherit from class Personal

```
class   Sub_Personal extend Personal
```

It will be illegal. You cannot derive from Personal class since it is a final class.

☞ Check Your Progress 3

1) Explain the advantage of abstract classes.
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

2) Write a program to show polymorphism in Java.
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

3) What are the different uses of final keyword in Java?

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

2.9 SUMMARY

In this unit the concept of inheritance is discussed. It is explained how to derive classes using extends keyword in Java. To control accessibility of data members three access specifiers: privatepublic, and protected-are discussed. How to write programs using concept of inheritance, method overriding, need of abstract classes is also explained. The concept of polymorphism is explained with the help of a programming example. In the last section of the unit the use of final keyword in controlling inheritance is discussed.

2.10 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Inheritance is used for providing reusability of pre-existing codes. For example, there is an existing class A and you need another class B which has class A properties as well as some additional properties. In this situation class B may inherit from class A and there is no need to redefine the properties common to class A and class B. In Java a class is inherited by any other class using *extends* keyword.

2) Access specifiers are used to control the accessibility of data members and member functions of class. It helps classes to prevent unwanted exposure of members (data and functions) to outside world.

3) If some data members of a class are used in internal operations only and there is no need to provide access of these members to outside world. Such member data should be declared private. Similarly, those member functions Which are used for internal communications/operations only should be declared private.

Check Your Progress 2

1)
   i. True
   ii. False
   iii. False

2) Java's super keyword is used for two purposes.

   i. To call the constructors of immediate superclass.
   ii. To access the members of immediate superclass.

When constructor is defined in any subclass it needs to initialize its superclass variables. In Java using **super()** superclass constructor is called **super()** must be the first executable statement in subclass constructor. Parameters needed by superclass constructor are passed in ***super (parameter_list)***. The super keyword helps in conflict resolution in subclasses in the situation of " when members name in superclass is *same as* members name in subclass and the members of the superclass to be called in subclass".

super.member; // member may be either member function or member data

3) Java uses Dynamic Method Dispatch, one of its powerful concepts to implement method overriding. Dynamic Method Dispatch helps in deciding the version of the method to be executed. In other words to identify the type of object on which method is invoked.

Overriding helps in:

- Redefining inherited methods in subclasses. In redefinition declaration should be identical, code may be different. It is like having another version of the same product.
- Can be used to add more functionality to a method.
- Sometimes class represent an abstract concept (i.e. abstract class). In this case it becomes essential to override methods in derived class of abstract class.

Check Your Progress 3

The advantage of abstract classes.

1) Any abstract class is used to provide a common interface to different classes derived from it. A common interface gives a feeling (understanding) of commonness in derived classes. All the derived classes override methods of abstract class with the same declaration. Abstract class helps to group several related classes together as subclass, which helps in keeping a program organized and understandable.

2)

```
// Program to show polymorphism in Java.
abstract class Account
{
public String Name;
public int Ac_No;
Account(String nm,int an )
{
Name=nm;
Ac_No= an;
}
abstract  void getAc_Info( );
}
class Saving_Account extends Account
{
private int min_bal;
Saving_Account( String na, int an, int bl)
{
super(na,an);
min_bal= bl;
}
void getAc_Info()
{
System.out.println(Name +"is having Account Number  :"+Ac_No);
System.out.println("Minimum Balance in Saving Account :"+Ac_No +"is
Rs:"+min_bal);
}
}
class Current_Account extends Account
{
private int min_bal;
Current_Account( String na, int an, int bl)
{
super(na,an);
min_bal= bl;
}
void getAc_Info()
{
System.out.println(Name +"is having Account Number  :"+Ac_No);
System.out.println("Minimum Balance in Current Account :"+Ac_No +"  is
Rs:"+min_bal);
}
}
public class AccountReference
{
public static void main(String[] args)
{
Account ref;                  // set up var for an Animal
Saving_Account s  = new Saving_Account("M.P.Mishra", 10001,1000);
Current_Account c  = new Current_Account("Naveen ", 10005,15000);
ref =s;
ref.getAc_Info();
ref =c;
ref.getAc_Info();
}
}
```

Output:

```
M.P.Mishrais having Account Number: 10001
Minimum Balance in Saving Account: 10001 is Rs: 1000
Naveen is having Account Number: 10005
Minimum Balance in Current Account: 10005 is Rs: 15000
```

3) Final keyword of Java is used for three things:

i. To declare a constant variable.
ii. To prevent a method to be overridden (to declare a method as final).
iii. To prevent a class to be inherited (to declare a class as final).