Static Class Members

- Static fields and static methods do not belong to a single instance of a class.
- To invoke a static method or use a static field, the class, rather than the instance name, is used.
- Example:

Static Methods

- Static methods are convenient because they may be called at the class level.
- they are typically used to create utility classes, such as the Math class in the Java standard Library.
- Static methods may not communicate with instance fields, only static fields.

Example:

```
public class Operation {
    public static int sum(int a,int b){
  return a+b;
    }
  public static int sub(int a,int b){
  return a-b;
    }
  public static int multi(int a,int b){
  return a*b;
    }
  public static int div(int a,int b){
  return a/b;
    }
```

```
public class Simpleoperation {
  public static void main(String[] args) {
    Operation.div(4, 2);
    Operation.sum(1, 3);
    Operation.sub(3, 4);
    Operation.multi(3, 4);
  }
```

Inheritance and Polymorphism

9.1 Introduction

Object-oriented programming allows you to derive new classes from existing classes. This is called *inheritance*. Inheritance is an important and powerful feature in Java for reusing software.

Suppose you are to define classes to model circles, rectangles, and triangles. These classes have many common features. What is the best way to design these classes so to avoid redundancy and make the system easy to understand and easy to maintain? The answer is to use inheritance.

Inheritance lets you create new classes from existing classes. Any new class that you create from an existing class is called a ***subClass*** or derived class; existing classes are called ***superclasses*** or base classes. The inheritance relationship enables a subClass to inherit features from its ***superclass***. Furthermore, the subClass can add new features of its own. Therefore, rather than create completely new classes from scratch, you can take advantage of inheritance and reduce software complexity.

Inheritance can be viewed as a hierarchical, structure where in a superclass is shown with its subClasses. Consider the diagram in Figure 9-1, which shows the relationship between various shapes.

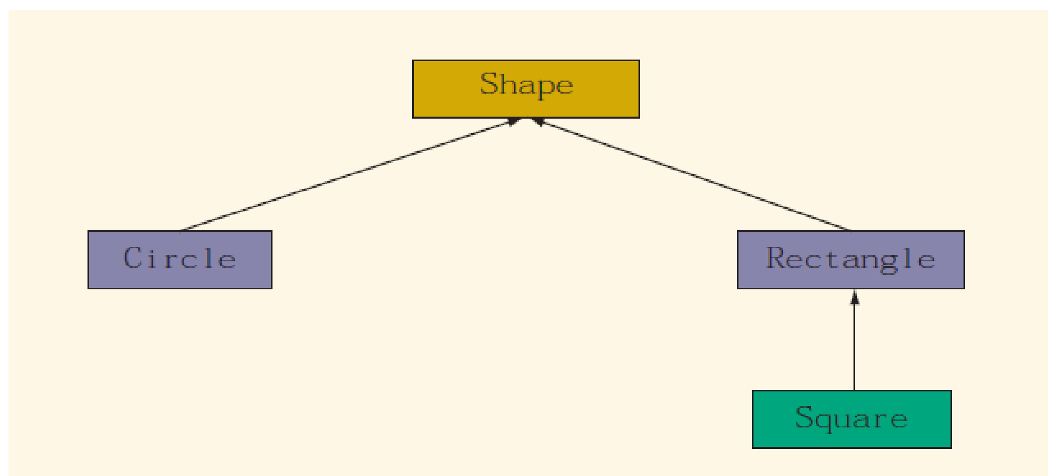


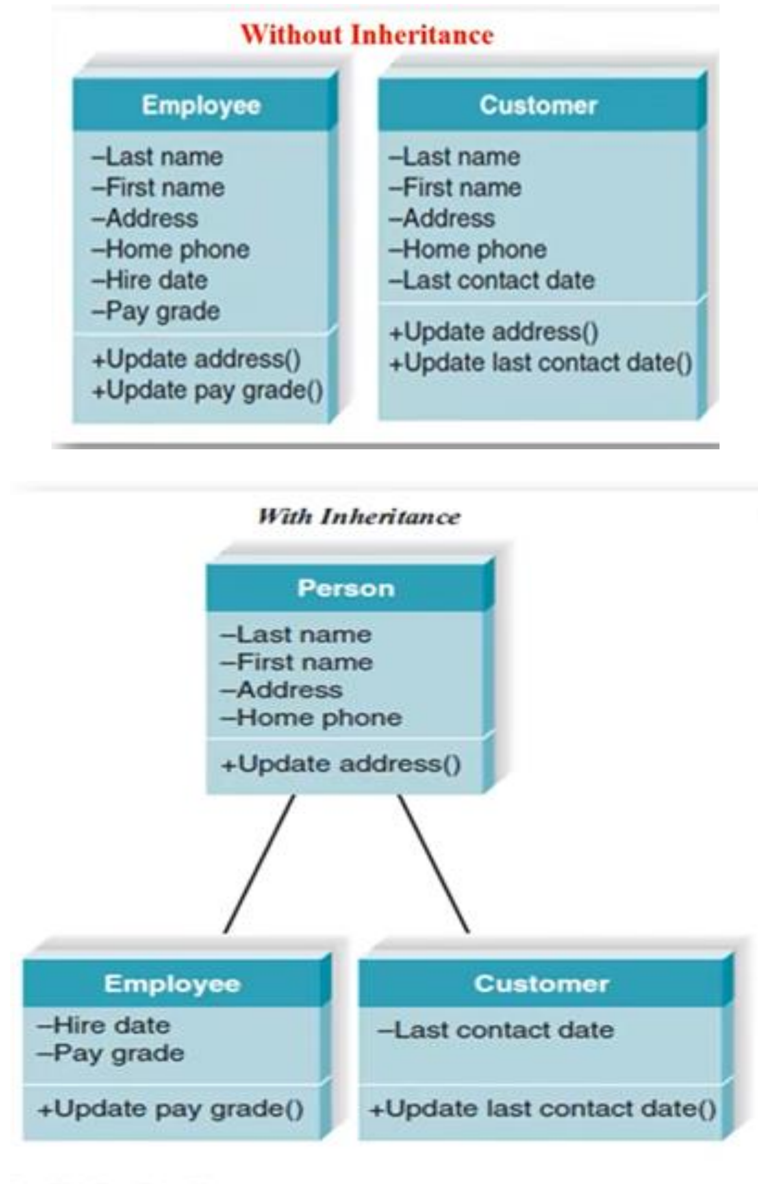
FIGURE 9-1 Inheritance hierarchy

In this diagram, **Shape** is the ***superclass***. The classes **Circle** and **Rectangle** are (***subClass***) derived from Shape, and the class Square is derived from Rectangle. Every Circle and every Rectangle is a Shape. Every Square is a Rectangle.

The general syntax to derive a class from an existing class is:

modifier(s) ***class*** ClassName ***extends*** ExistingClassName
{
memberList
}

In Java, ***extends*** is a reserved word.



Examples:-

SubClass **superClass**

- **public class child *extends* Father {**

}

- public class Circle extends Shape

```
{
.
.
.
}
```

- class Pet {

```
  private  String   name;
    public String   getName ( ) {
      return name;
    }
    public void setName( String   petName ) {
      name = petName;
    }
    public String speak (  ) { return
        "I'm your little pet.";
    }
  }


class Cat extends Pet {
        //This indicates Cat is  a subClass of superclass Pet
    public String speak( ) {
        return "Don't give me orders.\n" + "I speak only when I want to.";
     }
public static void main (string [ ] args) {
  setName("meo");
  System.out.println("Hi, my name is " +  getName( ));
 }
 }
```

We call the Cat class the ***subClass*** or *derived* class and the Pet class the ***superclass*** or *base* class. We use the reserved word *extends* to define a subClass. Data members and methods of a superclass are inherited by its suAClasses. so that we call (execute) ***setName*** and ***getName*** methods in Cat class directly, where it defined in Pet class

The following rules about superclasses and subClasses should be kept in mind:

1. ` The **private** members of the superclass are private to the superclass; hence, the members of the subClass(es) cannot access them directly. In other words, you cannot access the private members of the superclass directly.
2. The subClass can directly access the public members of the superclass.
3. The subClass can include additional data and/or method members.
4. The subClass can override, that is, redefine, the public methods of the superclass.
5. All data members of the superclass are also data members of the subClass. Similarly, the methods of the superclass (unless overridden) are also the methods of the subClass.
6. Each subClass, in turn, may become a superclass for a future subClass. Inheritance can be either single or multiple. In single inheritance, the subClass is derived from a single superclass; in multiple inheritance, the subClass is derived from more than one superclass.

Java supports only single inheritance; that is, in Java a class can extend the definition of only one class.

9.1 Overriding Methods

The subClass can give some of its methods the same signature as given by the superclass. For example, suppose that SuperClass contains a method, ***print***, that prints the values of the data members of SuperClass. SubClass contains data members in addition to the data members inherited from SuperClass. Suppose that you want to include a method in SuAClass that prints the data members of SuAClass. You can give any name to this method. However, in the class SuAClass, you can also name this method ***print*** (the same name used by SuperClass). This is called ***overriding***, or redefining, the method of the superclass.

The ***overriding*** means, you can have a method in the subClass with the same name, number, and types of parameters as a method in the superclass.

To override a public method of the superclass in the subClass, the method must be defined using the same signature and the same return type as in its superclass. If the corresponding method in the superclass and the subClass has the same name but different parameter lists, then this is method ***overloading*** in the subClass, which is also allowed.

Example:-

```
public class Test{
 int read( ){
   Scanner in = new Scanner (System.in);
  return in.nextInt( );
 }
```

```
    void print ( int x ){
      System.out.print(" these is superClass output" + x );
    }
}

  class overridingMethodTest  extends Test {

  String read(  ){
       Scanner in = new Scanner (System.in);
      return in.next( );
      }

    void print ( int x ){
      System.out.print(" these is subClass output" + x );
    }
   public static void main (String [ ] args) {int
     number = read ();
     print (number);
   }
}
```

In program up the method **print (int x)** are defined in both classes **Test** (superClass) and **overridingMethodTest** (subClass) with the same name, number, and types of parameters so that the **print (int x) is overridingMethod.**

The **read( )** method also defined in both classes but in different signature (differentreturn type) so that its overloading method

9.2 Overriding vs. Overloading

- Overloading means to define multiple methods with the same name but different signatures.

- Overriding means to provide a new implementation for a method in the subClass. The method is already defined in the superclass.

Let us use an example to show the differences between overriding and overloading. In (a) below, the method **p(double i)** in class **A** overrides the same method defined in class

B. In (b), however, the class **B** has two overloaded methods **p(double i)** and **p(int i)**.The method **p(double i)** is inherited from **B**.

```java
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

(a)

```java
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

(b)

9.3 Calling Superclass Methods

- If the subClass **overrides** a public method of the superclass, then you must specify a call to that public method of the superclass by using the reserved word ***super***, the general syntax to call a method of the superclass is:

  super.methodName (parameters);

- If the subClass does not override a public method of the superclass, you can specify a call to that public method by using just name of the method and an appropriate parameter list.

9.4 Constructors of the Superclass

A constructor typically serves to initialize the instance variables. When we instantiate a subClass object. The general syntax to call a constructor of a superclasss is:

super (parameters);

9.5 Protected Members of a Class

The private members of a class are private to the other class and cannot be directly accessed outside the class. Only methods of that class can access the private members directly. The subClass cannot access the private members of the superclass directly. However,

sometimes it may be necessary for a subClass to access a private member of a superclass. If you make a private member public, then anyone can access that member. Recall that the members of a class are classified into three categories: public, private, and protected. So, if a member of a superclass needs to be (directly) accessed in a subClass and yet still prevent its direct access outside the class, such as ina user program, you must declare that member using the modifier **protected**. Thus, the accessibility of a protected member of a class falls between public and private. A subClass can directly access the protected member of a superclass. To summarize, if a member of a superclass needs to be accessed directly (only) by a subClass, that member is declared using the modifier protected.

Example bellow illustrates how the methods of a subClass can directly access aprotected member of the superclass.

```
public class AClass {
  protected char    protectedCh;
  private     double privateX;
    //Default constructor
   public AClass() {
      protectedCh = '*' ;
      privateX = 0.0;
    }
//Constructor with parameters public
AClass(char ch, double u) {
   protectedCh = ch;
   privateX = u;
 }

public void setData(double u) {
  privateX = u;
 }

public void setData(char ch, double u) {
  protectedCh = ch;
  privateX = u;
}
public String toString( ) {
   return ("Superclass: protectedCh = " + protectedCh + ", privateX = "
   + privateX + '\n');
}
}
```

The definition of the class AClass contains the protected instance variable protectedCh of type char, and the private instance variable privateX of type double. It also contains an overloaded method setData; one version of setData is used to set both the instance variables, and the other version is used to set only the private instance variable. The class AClass also has a constructor with default parameters.

Next, we derive a class BClass from the class AClass. The class BClass contains a private instance variable dA of type int. It also contains a method setData, with three parameters, and the method toString.

```
public class BClass extends AClass {
   private int dA;

    public BClass( ) {
      super( );
      dA = 0;
}
public BClass(char ch, double v, int a) {
         super(ch, v);
        dA = a; public void setData(char ch, double v, int a) {
        super.setData(v);
        bCh = ch; //initialize bCh using the assignment statementdA =
a;
public String toString( ) {
      return (super.toString() + "Subclass dA = " + dA + '\n');
}
}
```

9.6 Protected Access vs Package Access

Typically a member of a class is declared with the modifier public, private, or protected to give appropriate access to that member.

- if a member of a class is declared public, then it can be directly accessed outside of the class.
- if a member of a class is declared private, then it cannot be directly accessed outside of the class.
- if a member is declared protected, it can be directly accessed in the class aswell as in any subclass.
- If a class member is declared without any of the modifiers public, private, or protected, then that member can be directly accessed in any class contained in same package.