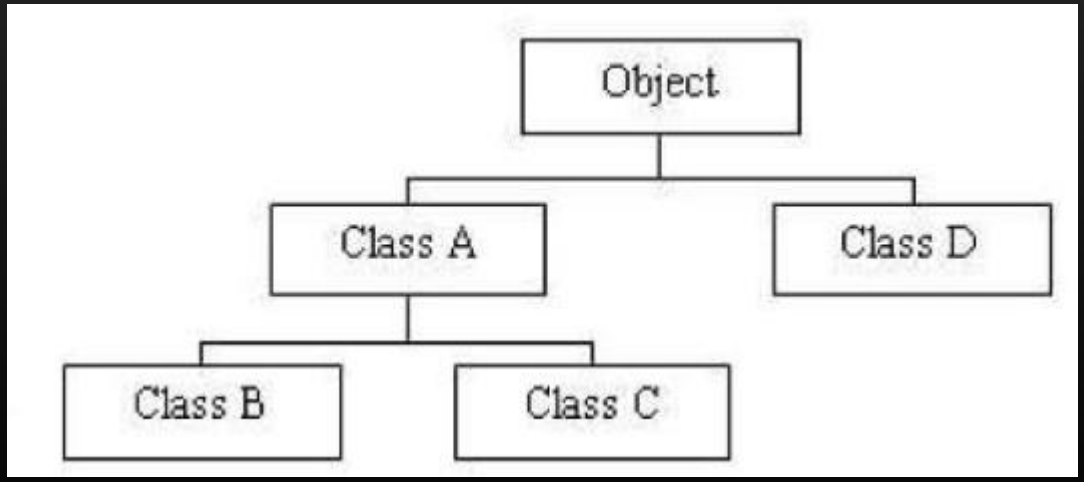
Inheritance

- the ability to create classes that share the attributes and methods of existing classes but with more specific features
- enables one class to acquire all the behaviors and attributes of another class



- Superclass/ base class/ parent
  - Any class above a specific class in the class hierarchy.
  - Common attributes
- Subclass/ derived class/ child
  - Any class below a specific class in the class hierarchy.
  - Class that inherits from a base class.
  - Unique attributes

Benefits of Inheritance

- Once a behavior (method) is defined in a superclass, that behaviour is automatically inherited by all subclasses.
- Thus, you can encode a method only once and they can be used by all subclasses.
- A subclass only needs to implement the differences between itself and the parent.

Extending classes

- To derive a class, use the extends keyword.

```
public class Person{
…
}
public class Student extends Person{
…
}
```

Overriding Superclass Methods

- To override a field or method in a child class means to use the child's version instead of the parent's version.
- Polymorphism - using the same method name to indicate different implementations
  - meaning "many forms"—many different forms of action take place, even though you use the same word to describe the action.

```
public class Person{
 public void display(){
  System.out.println(“Person Display Method”);
 }
}
public class Student extends Person{
 @Override
 public void display(){
  System.out.println(“Student Display Method”);
 }
}

public class DemoOvveriding {
  public static void main(String[] args){
    Student child = new Student();
    child.display();
  }
}
```

Output:
Student Display Method

Methods You Cannot Override

1. Static methods
   - A subclass cannot override methods that are declared static in the superclass
2. Final methods
   - A subclass cannot override methods that are declared final in the superclass
3. Methods within final classes
   - Cannot extend a final class

Calling Constructors During Inheritance

- When you create any object, as in the following statement, you are calling a constructor:
  - SomeClass anObject = new SomeClass();
- When you instantiate an object that is a member of a subclass, you are actually calling at least two constructors:
  - the constructor for the base class and the constructor for the extended, derived class.
- When you create any subclass object, the superclass constructor must execute first, and then the subclass constructor executes.

```
public class ASuperClass{
  public ASuperClass(){
    System.out.println("In superclass constructor");
  }
}

public class ASubClass extends ASuperClass{
  public ASubClass(){
    System.out.println("In subclass constructor");}
}

public class DemoConstructors{
  public static void main(String[] args){
    ASubClass child = new ASubClass();}
}
```

Output:
In superclass constructor
In subclass constructor

The “super” keyword

- A subclass can also explicitly call a constructor of its immediate superclass.
- This is done by using the super constructor call.
- A super constructor call in the constructor of a subclass will result in the execution of relevant constructor from the superclass, based on the arguments passed.
- Ex:

```
public Student(){
   super( "SomeName", "SomeAddress" );
   System.out.println("Inside Student:Constructor");
}
```

- The super() call MUST OCCUR AS THE FIRST STATEMENT IN A CONSTRUCTOR.
- The super() call can only be used in a constructor definition.
- This implies that the this() construct and the super() calls CANNOT BOTH OCCUR IN THE SAME CONSTRUCTOR.
- Another use of super is to refer to members of the superclass (just like the this reference).

Ex:
```
public Student() {
        super.name = “somename”;
        super.address = “some address”;}
```

Abstract class

- a class that cannot be instantiated
- often appears at the top of an object-oriented programming class hierarchy, defining the broad types of actions possible with objects of all subclasses of the class
- Include two method types:
  - Nonabstract methods, like those created in any class, are implemented in the abstract class and are simply inherited by its children
  - Abstract methods have no body and must be implemented in child classes

Abstract Methods

- Those methods in the abstract classes that do not have implementation are called abstract methods.
- To create an abstract method, just write the method declaration without the body and use the abstract keyword.
- Example,

```
public abstract void someMethod();
```

- When making abstract declarations:
  - If you declare a class to be abstract, each of its methods can be abstract or not.
  - If you declare a method to be abstract, you must also declare its class to be abstract.

```
//abstract class
public abstract class LivingThing{
 public void breath(){
  System.out.println("Living Thing breathing...");}
 //abstract method
 public abstract void walk();
}
```

When a class extends the LivingThing abstract class, it is required to override the abstract method walk(), or else, that subclass will also become an abstract class, and therefore cannot be instantiated.

```
public class Human extends LivingThing {
 public void walk(){
  System.out.println("Human walks...");}
}
```

```
public abstract class Card{
  public String recipient;
  public abstract void greeting();
}

public class Holiday extends Card{
  public Holiday( String r ){
    recipient = r;
  }
  public void greeting(){
    System.out.println("Dear " +
        recipient + ",\n");
    System.out.println(
        "Season's Greetings!\n\n");

  }
}
```

```
public class Birthday extends Card {
  public int age;
  public Birthday(String r, int years){
    recipient = r;
    age = years;
  }

  public void greeting(){
    System.out.println("Dear " +
        recipient + ",\n");
    System.out.println("Happy " + age +
        "th Birthday\n\n");
  }
}
```

```
public class Valentine extends Card {
  public int kisses;
  public Valentine(String r, int k){
    recipient = r;
    kisses = k;
  }

  public void greeting(){
    System.out.println("Dear " +
        recipient + ",\n");
    System.out.println(
        "Love and Kisses,\n");
    for ( int j=0; j<kisses; j++ )
      System.out.print("X");
    System.out.println("\n\n");
  }
}
```

```
import javax.swing.*;
public class Main {
  public static void main(String[] args)
  {
    String me;
    me = JOptionPane.showInputDialog(
      “Enter your name”);
    Holiday hol = new Holiday(me);
    hol.greeting();
    Birthday bd = new Birthday(me, 21);
    bd.greeting();
    Valentine val=new Valentine(me, 7);
    val.greeting();
  }
}
```

Using Parent Class Reference Variables

```
import javax.swing.*;
public class Main {
  public static void main(String[] args)
  {
    String me;
    me = JOptionPane.showInputDialog("Enter your name");
    Card card1 = new Holiday(me);
    card1.greeting();
    Card card2 = new Birthday(me, 21);
    card2.greeting();
    Card card3 = new Valentine(me, 7);
    card3.greeting();
  }
}
```

Polymorphism

- “having many forms”
- it means that a single variable might be used with several objects of related classes at different times in a program

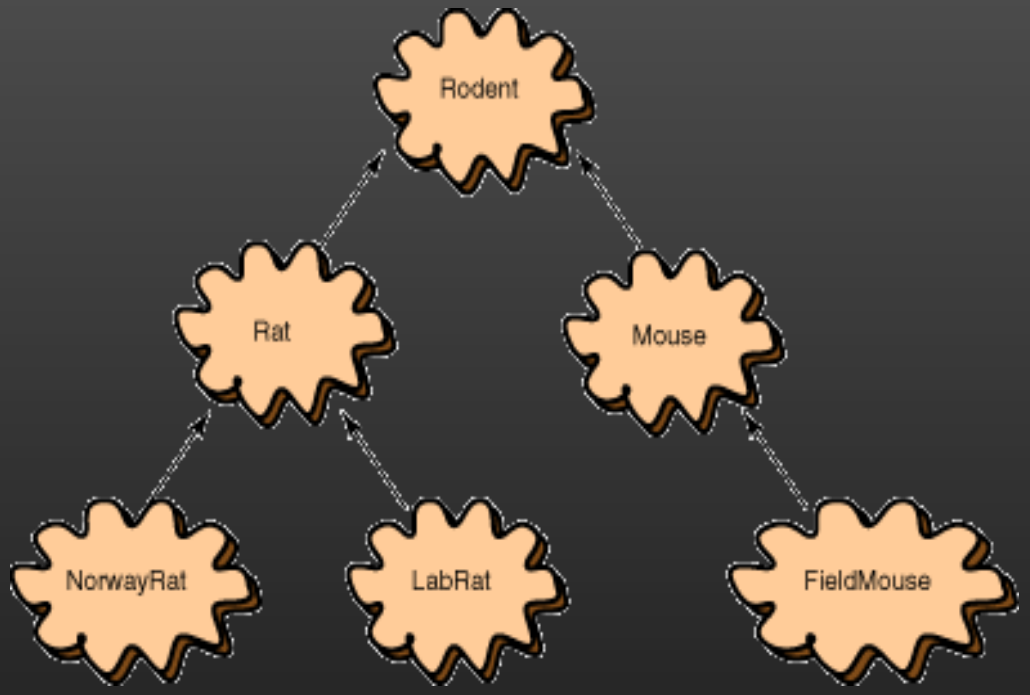
```
public class CardTester{
 public static void main(String[] args){

   Card card = new Holiday( "Amy" );
   //Invoke a Holiday greeting()
   card.greeting();

   card = new Valentine( "Bob", 3 );
   //Invoke a Valentine greeting()
   card.greeting();

   card = new Birthday( "Cindy", 17 );
   //Invoke a Birthday greeting()
   card.greeting();

 }
```



Here are some variables:

```
Rodent rod;
Rat rat;
Mouse mou;
```

code section	OK or Not?	code section	OK or Not?
rod = new Rat();		rod = new FieldMouse();	
mou = new Rat();		mou = new Rodent();	
rat = new Rodent();		rat = new LabRat();	
rat = new FieldMouse();		rat = new Mouse();	

Interfaces

- is a special kind of block containing method signatures (and possibly constants) only.
- defines the signatures of a set of methods, without the body.
- defines a standard and public way of specifying the behavior of classes.
- allows classes, regardless of their locations in the class hierarchy, to implement common behaviors.
- NOTE: interfaces exhibit polymorphism as well, since program may call an interface method, and the proper version of that method will be executed depending on the type of object passed to the interface method call.

Differences between an interface and an abstract class:

- In an interface, all methods do not have a body and you can only define constants. (an interface is not a class)
- For abstract classes, they are just like ordinary class declarations, with some abstract methods.
- Interfaces have no direct inherited relationship with any particular class, they are defined independently.

- To use an interface, use the keyword implements
- When your class tries to implement an interface, always make sure that you implement all the methods of that interface, or else, you would encounter an error message.

```
public interface Relation{
  public boolean isGreater(Object a,Object b);
  public boolean isLess(Object a,Object b);
}
public class Line implements Relation{
 public boolean isGreater(Object a, Object b){
  double aLen = ((Line)a).getLength();
  double bLen = ((Line)b).getLength();
  return (aLen > bLen);}
 public boolean isLess(Object a,Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen < bLen); }
}
```