Inheritance

2140101 Computer Programming for International Engineers

Objectives

Students should:

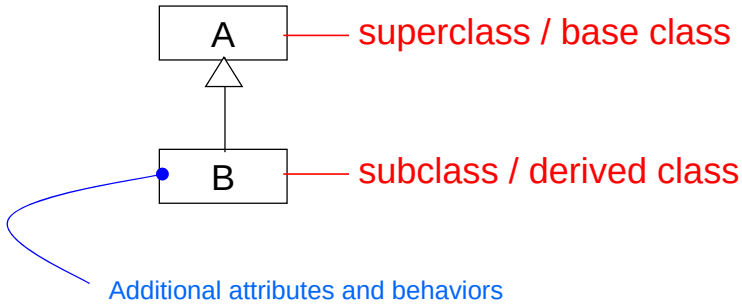
- Understand the concept and role of inheritance.
- Be able to design appropriate class inheritance hierarchies.
- Be able to make use of inheritance to create new Java classes.
- Understand the mechanism involved in instance creation of a class inherited from another class.
- Understand the mechanism involved in method invocation from a class inherited from another class.

Creating Subclasses from Superclasses

- Inheritance is an ability to derive a new class from an existing class.
- The new class is said to be a *subclass*, or *derived class*, of the class it is derived from, which is called *superclass*, or *base class*.
- A subclass can be thought of as an extension of its superclass. It inherits all attributes and behaviors from its superclass.
- More attributes and behaviors can be added to existing ones of its superclass.

Creating Subclasses from Superclasses

- Inheritance is an ability to derive a new class from an existing class.



Creating subclasses

- Suppose that we have a class called *L12A*

```
public class L12A
{
        public int x;
        public double d;
        public double f(){
                return x*d;
        }
}
```

- To create a new class L12B that inherits all attributes and behaviors from L12A, We use the keyword "extends" to let Java know that L12B inherits from L12A.

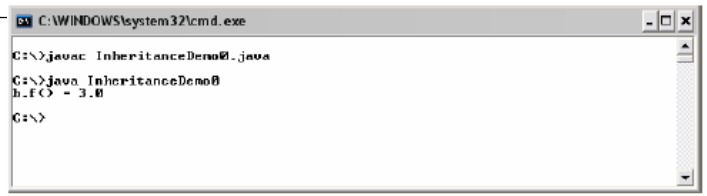
possible additional attributes/behaviors could be listed in here.

```
public class L12B extends L12A
{
}
```


Creating subclasses

- The following program that makes use of *L12B*.

```
public class InheritanceDemo0
{ public static void main(String[] args)
        {
                L12B b = new L12B();
                b.x = 2;
                b.d = 1.5;
                System.out.println("b.f() = "+b.f());
        }
}
```

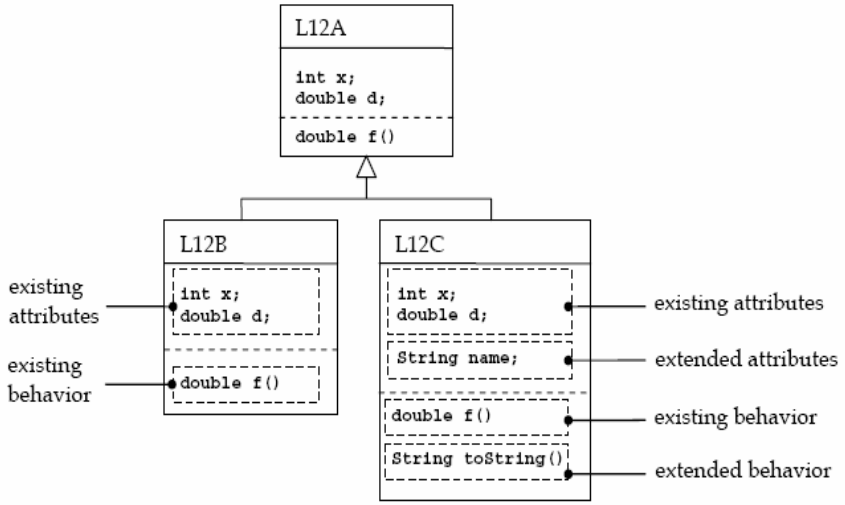


Creating subclasses

- Creating a new class contains all attributes and behaviors of its superclass together with some additional attributes and behaviors.

```
public class L12C extends L12A
{
        public String name;
        public String toString(){
                return name+":"+x+","+d;
        }
}
```


Creating subclasses



Example of creating subclasses

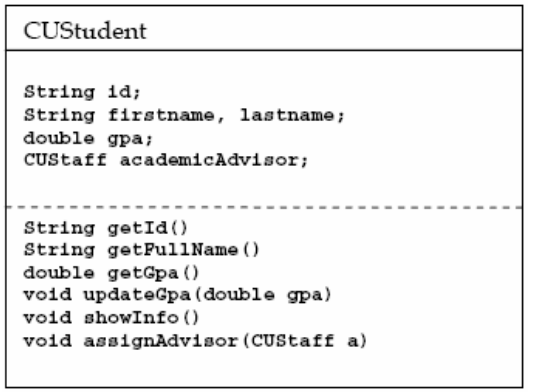
- consider the following program and observe its output.

```
C:\>javac InheritanceDemo01.java

C:\>java InheritanceDemo01
c.f() = 10.0
c.name = An object with a name
An object with a name:5.2.0

C:\>_
```

```
public class InheritanceDemo01
{
	public static void main(String[] args)
	{
		L12C c = new L12C();
		c.x = 5;
		c.d = 2.0;
		c.name = "An object with a name";
		System.out.println("c.f() = "+c.f());
		System.out.println("c.name = "+c.name);
		System.out.println(c);
	}
}
```

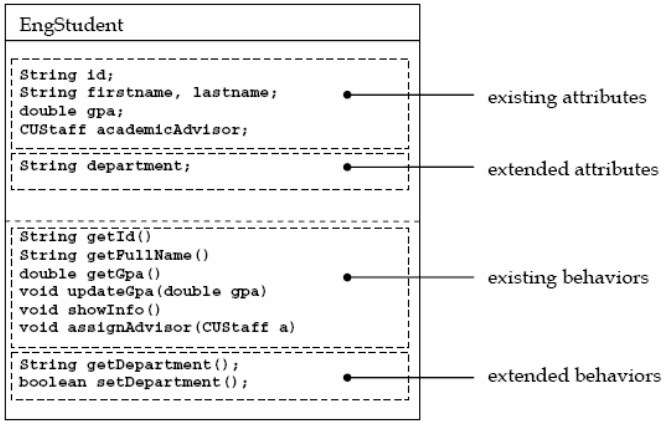

A More Realistic Example

- The superclass



A More Realistic Example

- The subclass



A More Realistic Example

```
public class CUStudent
{
    private String id;
    private String firstname, lastname;
    private double gpa;
    private CUStaff academicAdvisor;
    public CUStudent(String id, String firstname, String lastname) {
      this.id = id;
      this.firstname = firstname;
      this.lastname = lastname;
      gpa = 0.0;
      academicAdvisor = null;
    }
    public String getId(){
      return id;
    }
    public String getFullName(){
      return firstname+" "+lastname;
    }
// continue on the next page
```

A More Realistic Example

```
    public double getGpa(){
      return gpa;
    }
    public void updateGpa(double gpa){
      this.gpa = gpa;
    }
    public void showInfo(){
      System.out.println();
      System.out.println("ID:"+getId());
      System.out.println(getFullName());
      System.out.println("Advisor:"+getAdvisor());
    }
    public void assignAdvisor(CUStaff a){
      academicAdvisor = a;
    }
    public CUStaff getAdvisor(){
      return academicAdvisor;
    }
    public String toString(){
      return "CUStudent:"+getFullName();
    }
}
```


A More Realistic Example

```
public class EngStudent extends CUStudent
{
    private department;
    public EngStudent(String id,String firstname,String lastname){
      super(id,firstname,lastname);
      department = null;
    }
    public String getDepartment(){
      return department;
    }
    public boolean setDepartment(String dept){
      if(validDepartment(dept)){
        department = dept;
        return true;
      }else{
        return false;
      }
    }
    private boolean validDepartment(String dept){
      // This method returns true if the input String is a valid department
      // in Engineering school. … :
    }
}
```


A More Realistic Example

- Let's look at the following example to see *EngStudent* in action.

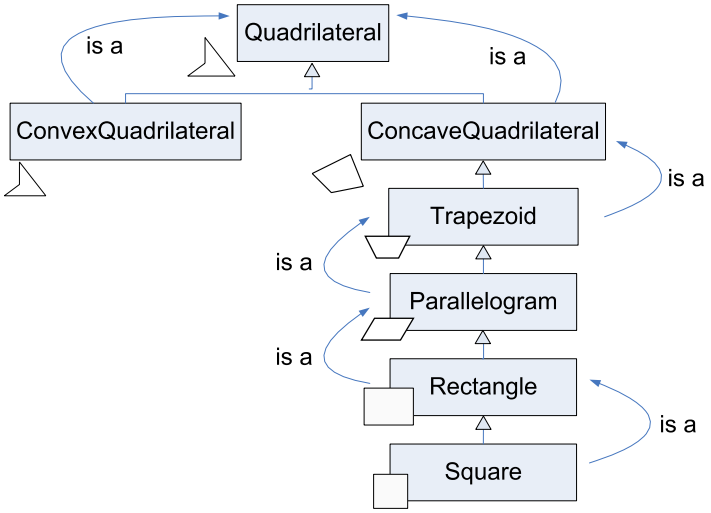
```
public class InheritanceDemo1
{
    public static void main(String[] args){
      EngStudent a = new EngStudent("4971234521","Alan","Smith");
      CUStaff b = new CUStaff("3600","Alex","Ferguson");
      a.assignAdvisor(b);
      a.showInfo();
      if(a.setDepartment("ICE"))
        System.out.println(a.getDepartment());
    }
}
```

```
C:\>javac InheritanceDemo1.java

C:\>java InheritanceDemo1

ID:4971234521
Alan Smith
Advisor:CUStaff:Alex Ferguson
ICE

C:\>
```


Designing Class Inheritance Hierarchy

- A good class hierarchy helps us understand the relationship among classes.
- Superclasses are always more general than subclasses since a subclass possesses everything that its superclass has while it can also possess additional attributes and behaviors.
- There is an *is-a relationship* between a subclass and its superclass.

Example of a hierarchy among some quadrilaterals.



Access Control

- Class access rights regarding to three access levels used in Java: **`public`**, **`private`**, and **`protected`**.
- When no explicit access levels are used, the access rights are the ones listed in the row labeled "default".

Access Level	Accessing Class		
	current class	subclass	other
public	☑	☑	☑
protected	☑	☑	☒
default	☑	☒	☒
private	☑	☒	☒

☑ : the corresponding accessing class can access resources with the corresponding access level

Access Control

```
public class MyDot2D
{
    private double x=0;
    private double y=0;
    public double getX(){
        return x;
    }
    public double getY(){
        return y;
    }
}
```


Access Control

```
public class AccessLevelDemo1
{
    public static void main(String[] args)
    {
      MyDot2D p = new MyDot2D();
      System.out.println(p.x);
      System.out.println(p.y);
    }
}
```

Compile: compilation errors since x and y have private access level and
attempts to access them directly using the dot operator

```
AccessLevelDemo1.Java:6: x has private access in MyDot2D
                         System.out.println(p.x);
AccessLevelDemo1.Java:7: y has private access in MyDot2D
                         System.out.println(p.y);
```

Keyword 'super'

```
public class MySuperclass
{
    public int a = 1;
    public int b = 2;

    public void f(){
      System.out.println("\tf() of MySuperclass is
            called.");
    }

    public void g(){
      System.out.println("\tg() of MySuperclass is
          called.");
    }
}
```


Keyword 'super'

```
public class MySubclass extends MySuperclass
{   public int a = 9;
    public void f(){
       System.out.println("\tf() of MySubclass is
       called.");
    }
    public void test(){
       System.out.println("a = "+a);
       System.out.println("b = "+b);
       System.out.println("super.a = "+super.a);
       System.out.println("super.b = "+super.b);
       System.out.println("f()");
       f();
       System.out.println("g()");
       g();
       System.out.println("super.f()");
       super.f();
       System.out.println("super.g()");
       super.g();
       }
}
```


Keyword 'super'

```
public class SuperDemo
{
    public static void main(String[] args)
    {
      MySubclass y = new MySubclass();
      y.test();
    }
}
```

```
C:\>javac  SuperDemo.java

C:\>java SuperDemo
a = 9
b = 2
super.a = 1
super.b = 2
f()
        f() of MySubclass is called.
g()
        g() of MySuperclass is called.
super.f()
        f() of MySuperclass is called.
super.g()
        g() of MySuperclass is called.
```


Object Variables

- A variable whose type is a class can refer to an object of that class as well as an object of any one of its subclasses.
- A variable of a class cannot be used to refer to an object of its superclass, just like when a variable cannot be used to refer an object whose class is different from the variable type.

Object Variables

- Recall the class hierarchy of *L12A*, *L12B* and *L12C*. Each of the following code segments are valid and compiled without errors.

```
L12A a = new L12A();
L12B b = new L12B();
L12C c = new L12C();
a = b;
a = c;
```

```
L12A a1 = new L12B();
L12A a2 = new L12C();
```

```
L12A [] a = new L12A[3];
a[0] = new L12A();
a[1] = new L12B();
a[3] = new L12C();
```


Object Variables

- However, the following program cannot be compiled successfully.

```
1  public class ObjectVariableInvalidDemo
2  {
3      public static void main(String[] args)
4      {
5        L12A a = new L12A();
6        L12B b = new L12B();
7        L12C c = new L12C();
8        b = a;
9        c = a;
10       b = c;
11     }
12 }
```

Incompatible types

Method Overriding and Polymorphism

- When a method that has already been defined in a class is redefined in its subclass using the same identifier and the same list of input arguments, it is called that the method defined in the subclass `overrides` the one in its superclass.
- When this happens, which method to be invoked, i.e. the one in the superclass or the one in the subclass, depends on the type of the object from which the method is invoked.

Method Overriding and Polymorphism

```
1  public class MethodOverridingDemo1
2  {
3      public static void main(String[] args)
4      {
5        L12A a = new L12A();
6        a.x = 2;
7        a.d = 1.0;
8
9        L12D d = new L12D();
10       d.x = 2;
11       d.d = 1.0;
12       d.y = 2.5;
13
14       System.out.println("a.f()="+a.f());
15       System.out.println("d.f()="+d.f());
16
17       a = d;
18       System.out.println("a.f()="+a.f());
19     }
20 }
```

```
C:\>javac MethodOverridingDemo1.java

C:\>java MethodOverridingDemo1
a.f()=2.0
d.f()=4.5
a.f()=4.5

C:\>_
```

Instance Creation Mechanism

- When an instance or object of a class is created, if there is no explicit call of any constructors of its superclass, the no-argument constructor of the superclass is called automatically before the execution of any statements in the subclass's constructor (if there are any).

Instance Creation Mechanism

```
public class L12E
{
    public L12E(){
       System.out.println("\tL12E() is called.");
    }
}
```

```
public class L12F extends L12E
{
}
```


Instance Creation Mechanism

```
public class L12G extends L12E
{
    public L12G(){
       System.out.println("\tL12G is called.");
    }
    public L12G(String s){
       System.out.println("\tL12G(String s) is called.");
    }
    public L12G(int i){
       super();
        System.out.println("\tL12G(int i) is called.");
    }
}
```

- L12F and L12G are subclasses of L12E. Let's consider the following program. Pay attention to messages printed on screen when each object is created.

Instance Creation Mechanism

```
1  public class CreationDemo1
2  {
3      public static void main(String[] args)
4      {
5          System.out.println("1)-----------------");
6          L12F f = new L12F();
7          System.out.println("2)-----------------");
8          L12G g1 = new L12G();
9          System.out.println("3)-----------------");
10         L12G g2 = new L12G("Hello");
11         System.out.println("4)-----------------");
12         L12G g3 = new L12G(8);
13     }
14 }
```

```
C:\>javac CreationDemo1.java

C:\>java CreationDemo1
1)-----------------
        L12E() is called.
2)-----------------
        L12E() is called.
        L12G is called.
3)-----------------
        L12E() is called.
        L12G(String s) is called.
4)-----------------
        L12E() is called.
        L12G(int i) is called.
```

Instance Creation Mechanism

- Keep in mind that the `super()` statement has to be used in the first statement only. Otherwise, the class will not be compiled successfully.

```
public class L12H extends L12E
{
    public L12H(){
       System.out.println("\tL12H is called.");
       super();
    }
}
```

```
C:\>javac L12H.java
L12H.java:5: call to super must be first statement in constructor
                super();
                ^
1 error

C:\>_
```