Chapter 10

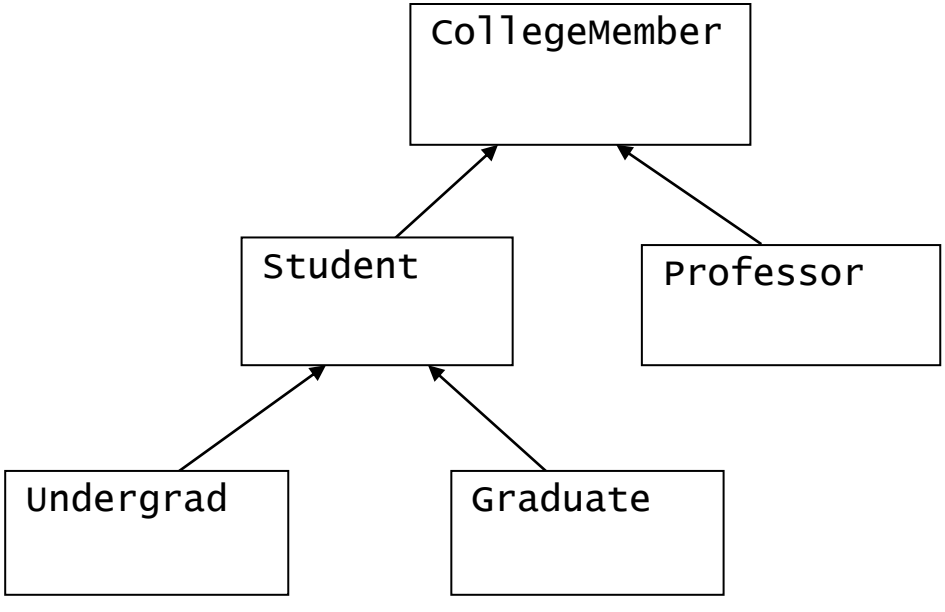
Inheritance Part 1

Advantages of Inheritance

1) Many real-world systems form a hierarchy of categories.

2) Inheritance allows the compiled form of classes to be modified and/or extended.

3) Inheritance eliminates the need to duplicate code.


CollegeMember
Student
Professor
Undergrad
Graduate

Superclass

```
 1 class A     // This is the superclass
 2 {
 3    private int x = 1;
 4    //-----------------------------------
 5    public int xGet()
 6    {
 7       return x;
 8    }
 9    //-----------------------------------
10    public String toString()
11    {
12       return("x = " + x);
13    }
14 }
```

Subclass

```
class B extends A
{
   int y = 7;
   //-----------------------------------
   public int yGet()
   {
      return y;
   }
   //-----------------------------------
   public String toString()
   {
      return "x = " + xGet() + " y = " + y;
   }
}
```

```
class A
{
  ...
}
```

Superclass (also known as base class)

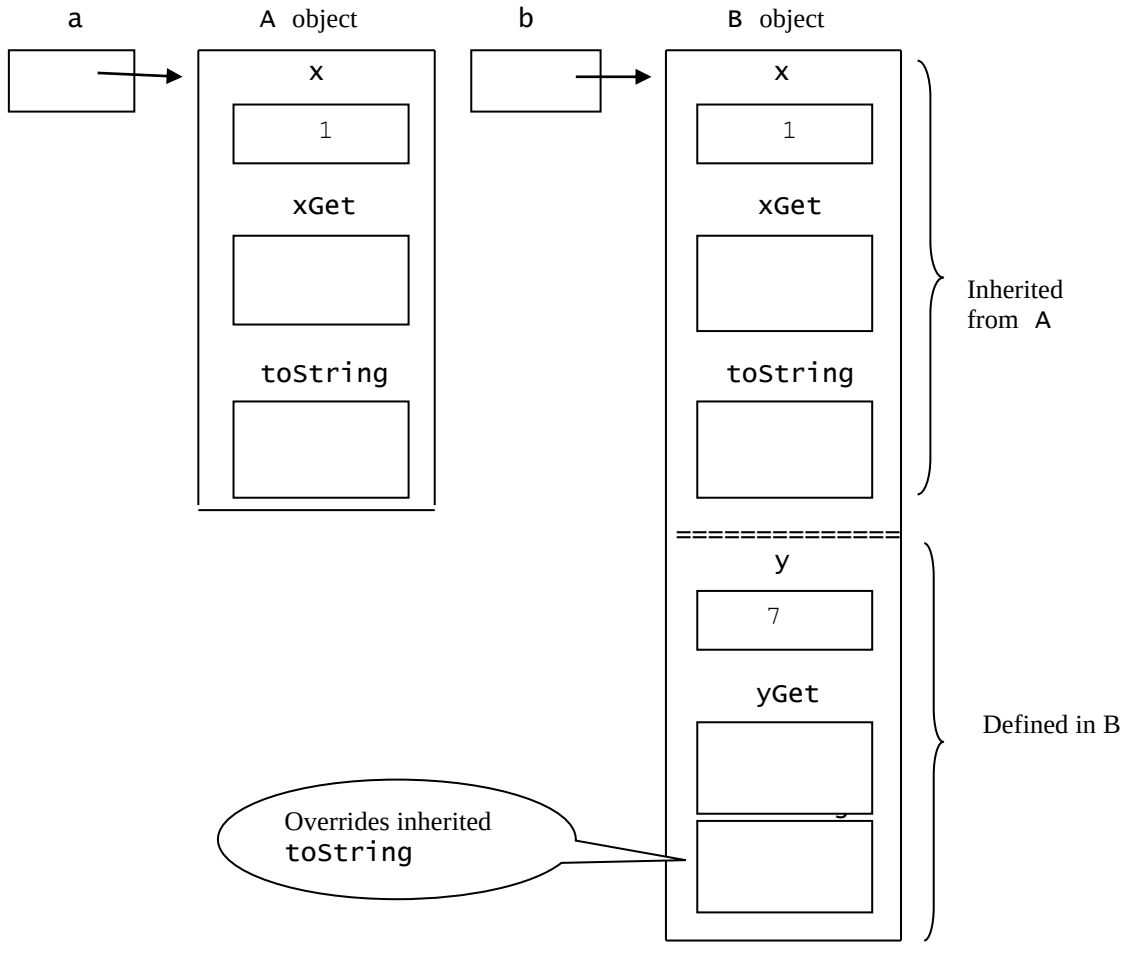
Arrow points from subclass to superclass

```
class B extends A
{
  ...
}
```

Subclass (also know as derived class)

```
A a = new A();
System.out.println(a.xGet());
System.out.println(a.toString());

B b = new B();
System.out.println(b.yGet());
System.out.println(b.xGet());
System.out.println(b.toString());
```


a
A object
b
B object
x
1
xGet
toString
x
1
xGet
toString
Inherited from A
==========
y
7
yGet
Defined in B
Overrides inherited toString

Constructors and inheritance

1) Every class has at least one constructor.

2) Every subclass constructor calls a constructor in its superclass.

```
16 class B extends A
17 {
18    private int y;
19    //----------------------------------
20    public B()
21    {
22       y = 2;
23    }
24    //------------------
25    public B(int yy)
26    {
27       super();         // explicit call of superclass constructor
28       y = yy;
29    }
30    //----------------------------------
31    public void xyDisplay()
32    {
33       xDisplay();      // call inherited method
34       System.out.println("y = " + y);
35    }
36 }
```

super();

inserted here

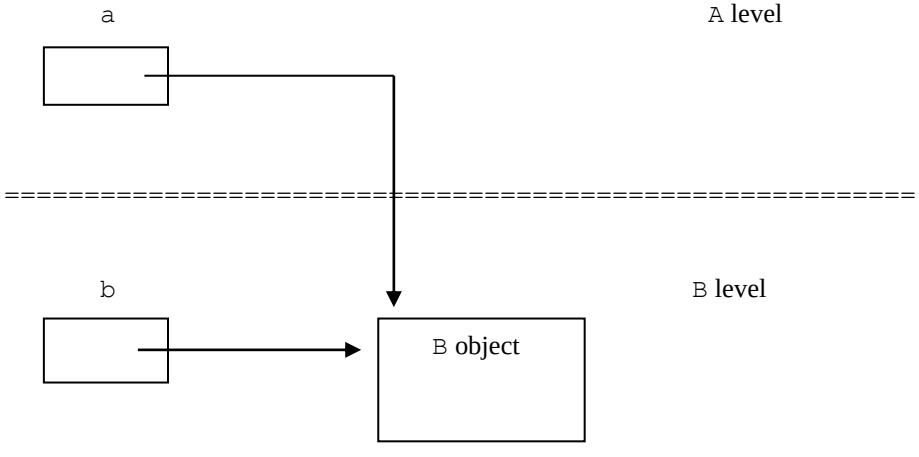
```
37 //=========================================
38 class C extends B
39 {
40    private int z = 4;
41    //-----------------------------------
42    public void xyzDisplay()
43    {
44       xyDisplay();
45       System.out.println("z = " + z);
46    }
47 }
48 //==============================================
49 class Inheritance2
50 {
51    public static void main(String[] args)
52    {
53       B b = new B();
54       b.xyDisplay();    // displays x = 1 y = 2
55       b = new B(3);
56       b.xyDisplay();    // displays x = 1 y = 3
57       C c = new C();
58       c.xyzDisplay();   // displays x = 1 y = 2 z =
4
59    }
60 }
```

Default constructor inserted here

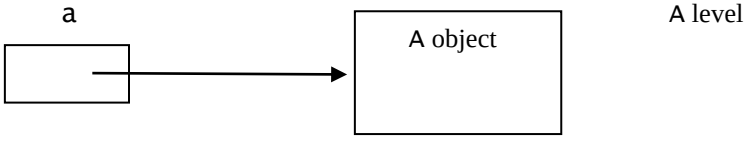
A toolbox and a toolbox with a cheese sandwich

```
A a = new A();
B b = new B();
```

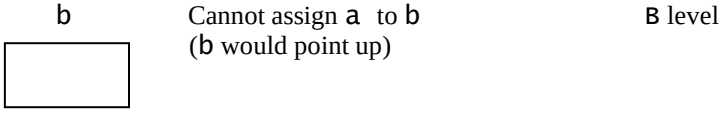
```
A a;
B b = new B();   // B is a subclass of A
a = b;           // legal
```



```
B b;                    // B is a subclass of A
A a = new A();
b = a;                  // illegal
```

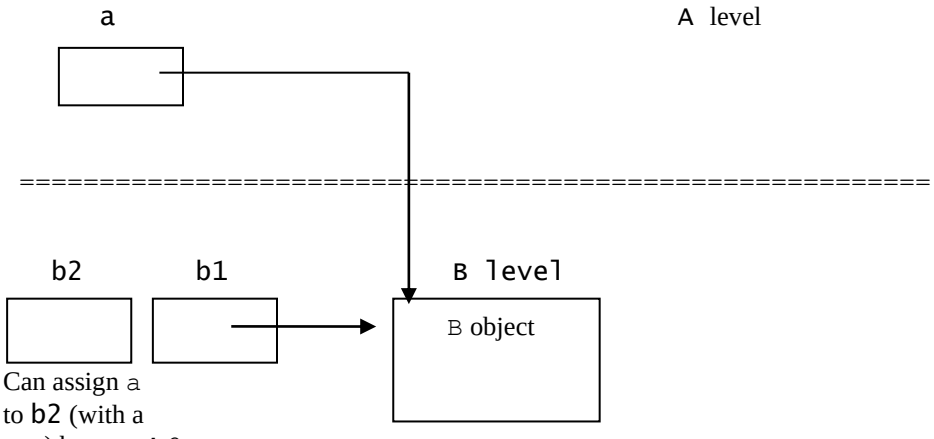


==========




```
b = (B)a;        // compiles ok but still illegal
```

```
A a;
B b1, b2;          // B is a subclass of A
b1 = new B();
a = b1;            // okay because a is pointing "down"
```



Can assign a to b2 (with a cast) because b2 would *not* point up.

```
b2 = a;          // illegal
b2 = (B)a;       // legal because a is pointing down
```

Downcasting and upcasting

```
A a
B b1, b2;
b1 = new B();
a = b1;          // upcast is legal
b2 = (B)a;       // this downcast is legal
```

A reference can be downcast if it was previously upcast.

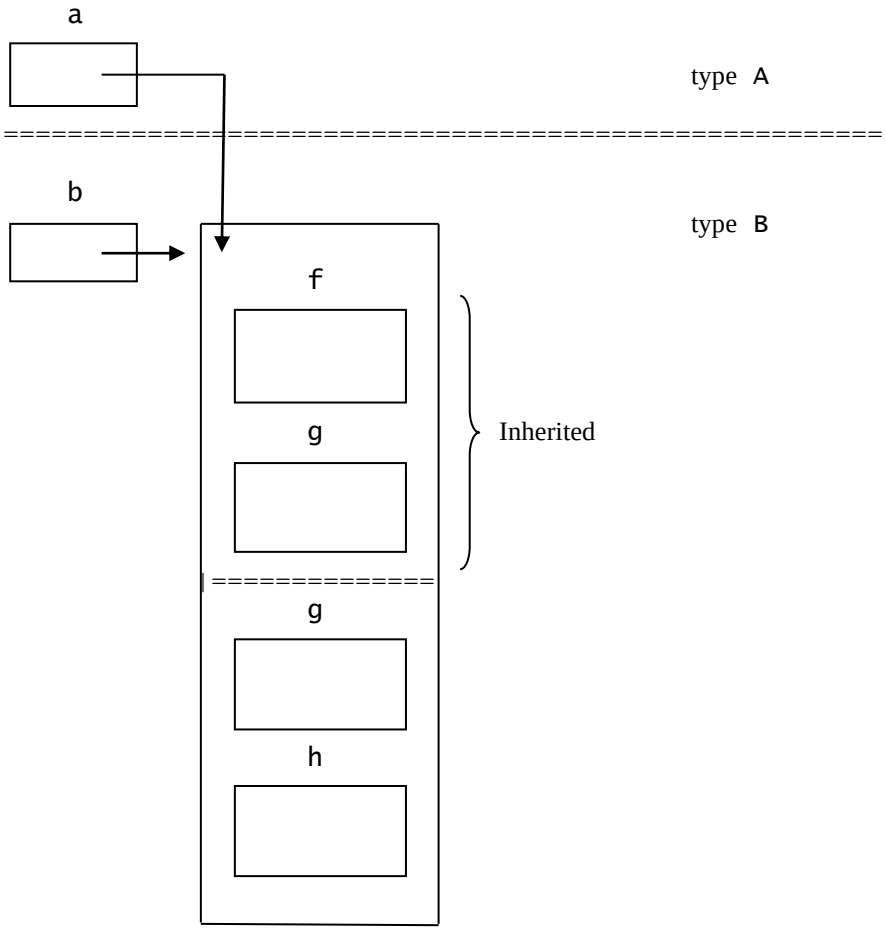
```
// check with instanceof
B b;
if (a instanceof
of B)
   b = (B)a;
```

Case study

```
class A
{
   public void f()
   {
      System.out.println("f in A");
   }
   //-----------------------------------
   public void g()
   {
      System.out.println("g in A");
   }
}
```

```
14 class B extends A
15 {
16    public void g()
17    {
18       System.out.println("g in B");
19    }
20    //-------------------------------------
21    public void h()
22    {
23       System.out.println("h in B");
24    }
25 }
```

```
27 class Inheritance3
28 {
29    public static void main(String[] args)
30    {
31       A a;
32       B b = new B();
33       a = b;            // upcast
34       a.f();            // displays "f in A"
35       a.g();            // displays "g in B"
36       a.h();    // illegal: compile-time error
37       ((B)a).h();       // displays "h in B"
38       b.h();            // displays "h in B"
39    }
40 }
```

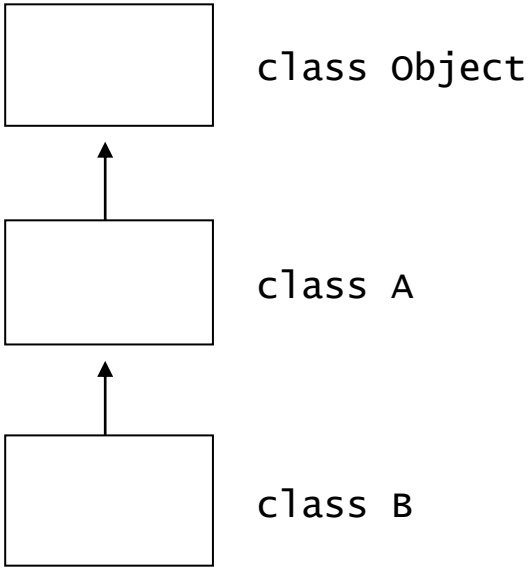

a
type A
b
type B
f
g
Inherited
g
h

`Object` class

```
class A
{
   ...
}
```

equivalent to

```
class A extends Object
{
   ...
}
```


class Object
class A
class B

toString and equals from Object

```
1 class A   // inherits toString and equals from Object
2 {
3    private int x = 1;
4 }
5 //==================================================
6 class B extends A // inherits from A
7 {
8    private int y = 2;
9 }
```

```
11 class Inheritance4
12 {
13    public static void main(String[] args)
14    {
15       A a1, a2;
16       a1 = new A();
17       a2 = new A();
18       System.out.println(a1.toString()); // A@3e25a5
19       System.out.println(a1.equals(a2)); // false
20
21       B b1, b2;
22       b1 = new B();
23       b2 = new B();
24       System.out.println(b1.toString()); // B@19821f
25       System.out.println(b1.equals(b2)); // false
26    }
27 }
```

Custom `toString`

```
1 class A  // inherits toString and equals
2 {
3    protected int x = 1; // NOTE!!
4    //-----------------------------------
5    public String toString()
6    {
7       return "x = " + x;
8    }
9 }
```

```
11 class B extends A
12 {
13    private int y = 2;
14    //-----------------------------------
15    public String toString()
16    {
17       return "x = " + x + " y = " + y;
18    }
19 }
```

```
21 class Inheritance5
22 {
23    public static void main(String[] args)
24    {
25       A a;
26       a = new A();
27       System.out.println(a.toString()); // x = 1
28
29       B b;
30       b = new B();
31       System.out.println(b.toString());//x = 1 y = 2
32    }
33 }
```

Another `toString`

```
class A
{
   private int x = 1;  // now private
   //-----------------------------------
   public String toString()
   {
      return "x = " + x;
   }
}
```

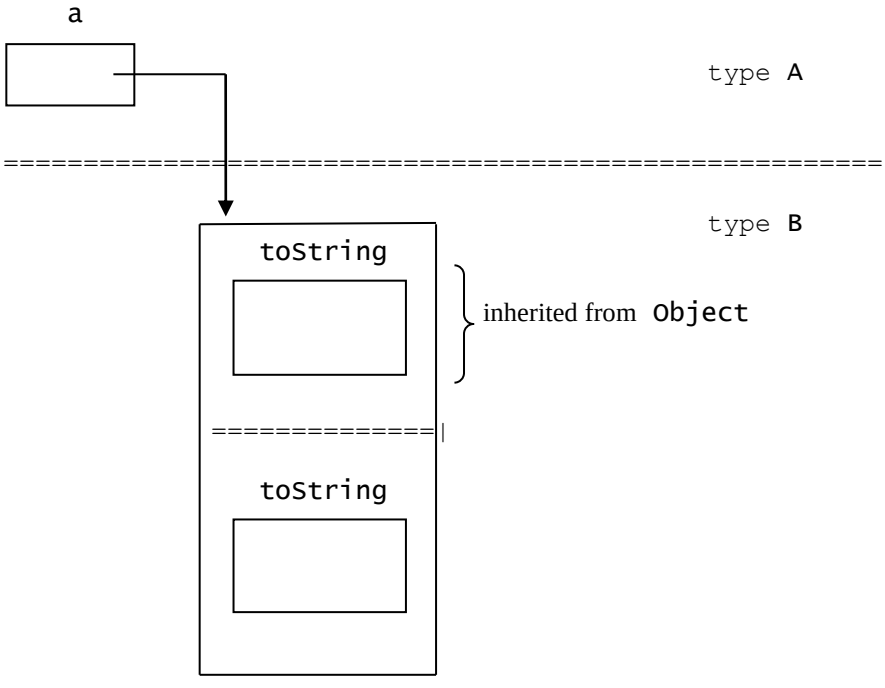
```
11 class B extends A
12 {
13    private int y = 2;
14    //-----------------------------------
15    public String toString()
16    {
17       return super.toString() + " y = " + y;
18    }
19 }
```

Another `toString`

```
 1 class A
 2 {
 3    private int x = 1;
 4    //-----------------------------------
 5    public String toString()
 6    {
 7       return "x = " + x;
 8    }
 9    //-----------------------------------
10    public int xGet()
11    {
12       return x;
13    }
14 }
```

```
16 class B extends A
17 {
18    private int y = 2;
19    //-----------------------------------
20    public String toString()
21    {
22       return  "x = " + xGet() + " y = " + y;
23    }
24 }
```

Why are `toString` and `equals` in the `Object` class?



Access Specifiers

```
class Specifiers
{
   private int w;     // private access
   int x;             // package access
   protected int y;   // protected
   public z;          // public access
   ...
}
```

Class can have public or package access

```
public class C
{
    ...
}


class C
{
    ...
}
```