

Inheritance



What is inheritance?

Inheritance

A Mammal is an Animal.

A Dog is a Mammal.

Old Yeller is a Dog.

A Bird is an Animal.

A Chicken is a Bird.

Foghorn Leghorn is a Chicken.

***x is a Y* – X is an extension of Y**

Inheritance

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

Inheritance essentially copies all of the methods and instance variables from class A and pastes those into class B at run time. The code from A is run from within class B.

There is way more to it than just a simple copy/paste, but the copy/paste analogy explains it well enough.

Inheritance

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

A class can extend one other class.

Java does not support multiple inheritance.

```
class C extends A,B { } //illegal
```

inheritance

example

```
class A{  
    private int x;  
    public A() { x =8;}  
    public String toString() {  
        return ""+x;  
    }  
}
```

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

```
//test code in the main method  
A one = new A();  
out.println(one);  
one = new B();  
out.println(one);
```

OUTPUT

8

8

inheritance

example

```
class A{  
    private int x;  
    public A() { x =3;}  
    public void setX(int val){  
        x=val;  
    }  
    public int getX(){ return x; }  
}
```

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

//test code in the main method

```
B one = new B();  
out.println(one.getX());  
one.setX(2);  
out.println(one.getX());
```

OUTPUT

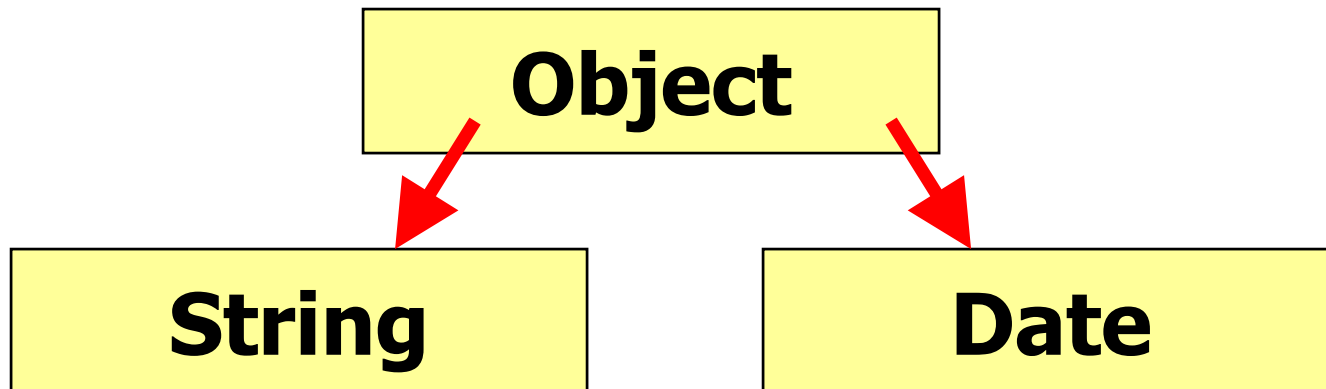
3

2

Open
inheritone.java

class Object

Class Object is the one true super class. Object does not extend any other class. All classes extend Object.



class Object

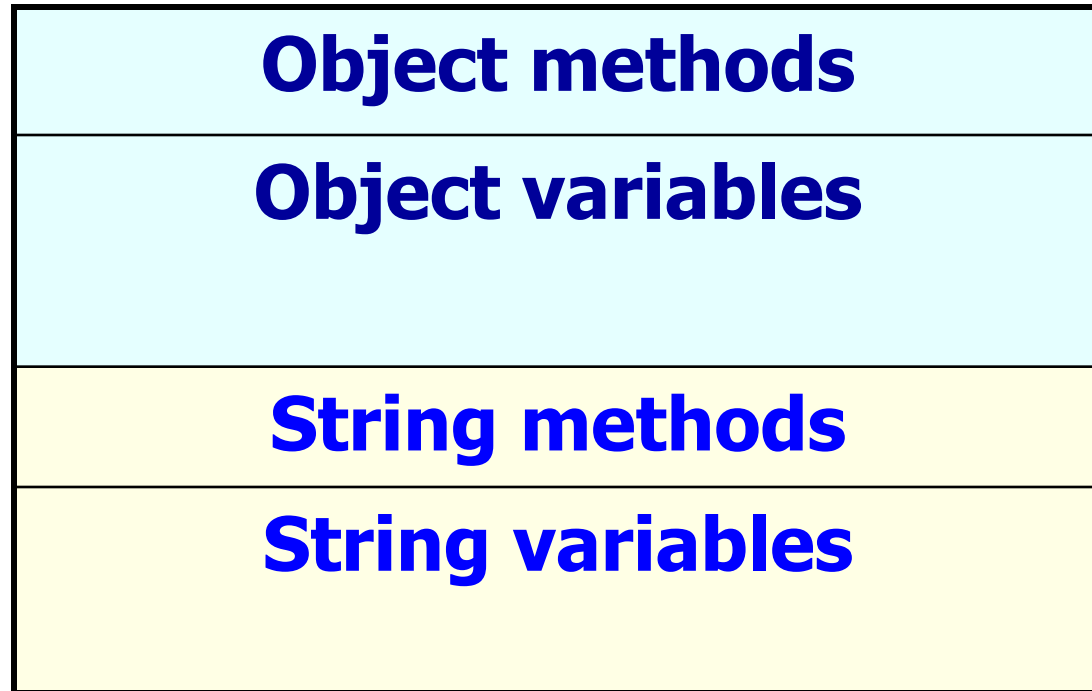
Because all classes are sub classes of Object, all Java classes start with at least the methods from Object.

```
.equals( )  
.toString( )  
.hashCode()  
.clone( )  
. . . . and more
```



What's on the inside?

**A String
Object**



A String is an Object!!

inheritance

example

two

```
class Monster
{
    private String myName;

    public Monster() {
        myName = "Monster";
    }
    public Monster( String name ) {
        myName = name;
    }
    public String toString() {
        return "Monster name :: " + myName + "\n";
    }
}

class Witch extends Monster
{
}
```

Open
inherittwo.java

Public

Protected

Private

public

All members defined as public can be accessed by members of the super class, sub class, or any other class.

protected

All members defined as protected can be accessed by members of the super class and sub class and any other class in the same package.

Protected is commonly referred to as package public.

private

All members defined as private can only be accessed by members of the class where they are defined.

Private members may not be accessed directly by sub classes or by other classes.

information hiding

Information hiding is a big part of good design. Information hiding is demonstrated with inheritance in that super class code is written, tested, and then tucked away. Sub classes can be written using the super class methods with no real concern for the implementation details in the super class methods.

Open

pubprotpriv.java

this

this – refers to the object/class
you are working in

this.toString(); calls the toString of this class

this.x = 1524;

this(); calls a constructor of this class

```
class Monster
{
    private String myName;
```

this

```
    public Monster() {
        this("Monster");
    }
```

calls Monster(name)

```
    public Monster( String name ) {
        myName = name;
    }
```

```
    public String toString() {
        return myName + "\n";
    }
}
```

**Open
this.java**

super

super – refers to the parent class

super.toString(); legal

super.super.toString(); illegal

super(); parent default constructor call

super("elmo", 6); parent constructor call

```
class Skeleton extends Monster  
{
```

```
    private double speed;
```

```
    public Skeleton( ) {
```

```
        speed=100;
```

```
    }
```

A super call is always made on the 1st line of any sub class constructor.

```
    public Skeleton( String name, double speed ) {
```

```
        super(name);
```

```
        this.speed=speed;
```

```
    }
```

super – refers to the parent

```
    public String toString( ) {
```

```
        return super.toString() + " " + speed;
```

```
    }
```

```
}
```

super this

Open

superthis.java

Start work on Lab 02

What's on the inside?

```
class Monster
```

```
{  
    private String myName = "long way to go for a toString()";  
    public Monster() { }  
    public Monster( String name ) { myName = name; }  
    public String toString( ) { return myName; }  
}
```

```
class Witch extends Monster
```

```
{  
    public Witch( ) { } //this constructor must exist  
    public Witch( String name ) { //automatically calls super( ) }  
}
```

```
class GoodWitch extends Witch
```

```
{  
    public GoodWitch() { //automatically calls super( ) }  
}
```

What's on the inside?

**GoodWitch
object**



Object methods
Object variables
Monster methods
Monster variables
Witch methods
Witch variables
GoodWitch methods
GoodWitch variables

Open

whatsontheinside.java

**Create 2 new monsters
from the Monster class.**

Polymorphism

Polymorphism - the ability of one general thing to behave like other specific things.

Polymorphism

Object x = "compsci";

System.out.println(x);

**Why is it okay to have an
Object refer to a String?**

OUTPUT
compsci

Polymorphism

Object x = "compsci";

System.out.println(x.toString());

**Why is it okay to call the
toString() method on x?**

OUTPUT
compsci

Polymorphism

Object x = "compsci";

System.out.println(x.length());

**Why is it not okay to call
the length() method on x?**

OUTPUT

syntax error

Polymorphism

Object x = "compsci";

out.println(((String)x).length());

**The cast will now let this
code compile.**

OUTPUT

7

Polymorphism

Witch x = new Monster();

System.out.println(x);

Is this okay or not okay?

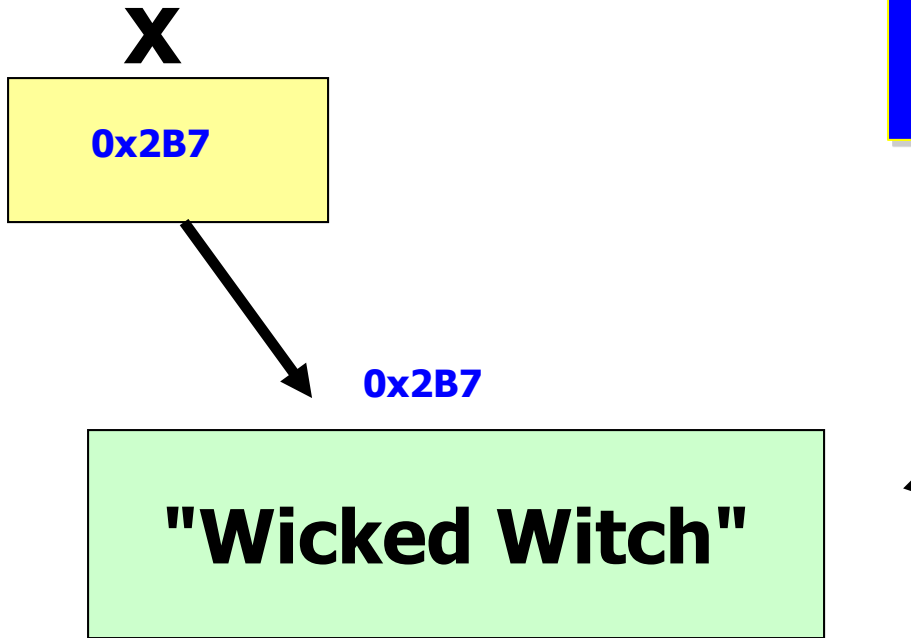
Polymorphism

```
Monster x = new Witch();  
Monster y = new Ghost();
```

```
System.out.println(x);  
System.out.println(y);
```

Is this okay or not okay?

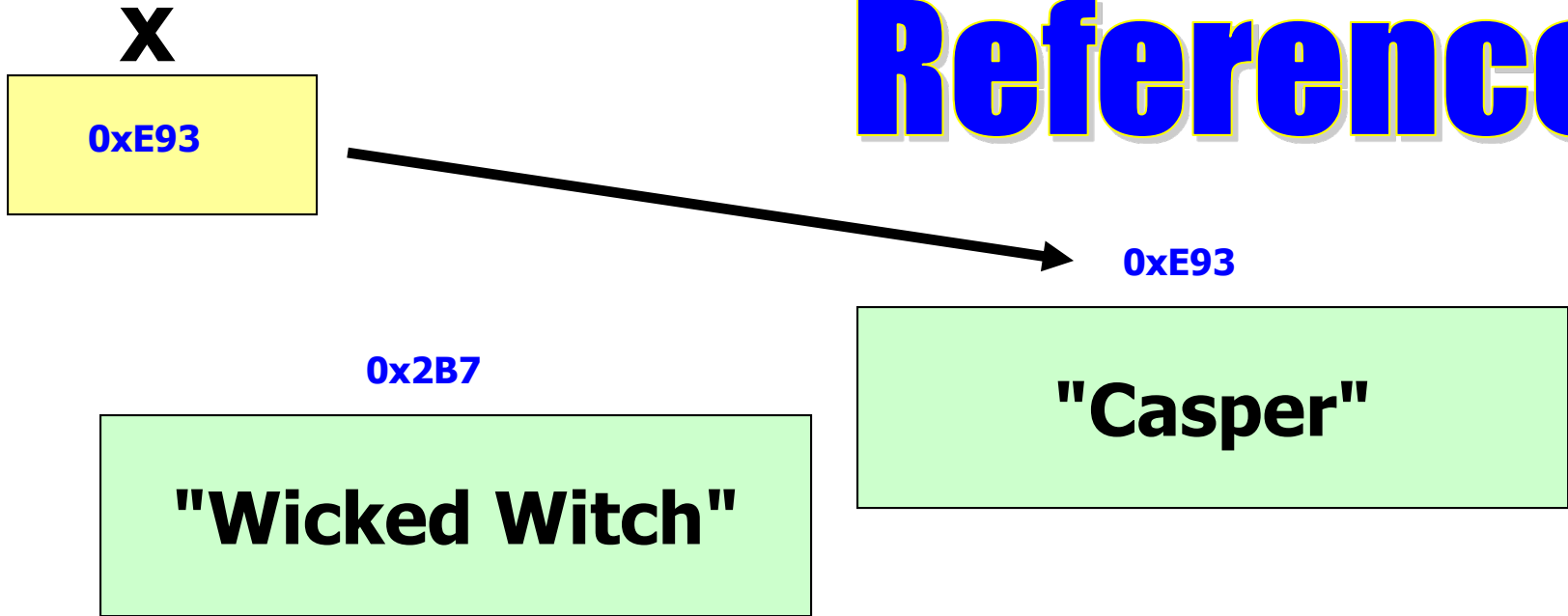
References



Monster x = new Witch("Wicked Witch");

Monster reference x refers to a Witch!

References



```
x = new Ghost("Casper");
```

Monster reference x now refers to a Ghost!

Open poly.java

method override

When you extend a class, you inherit all methods and instance variables.

You can override the original methods by implementing one with the same signature.

method override

```
class Monster
{
    private String myName;
    public Monster( String name ) {
        myName = name;
    }
    public void overRide( ) {
        System.out.println("overRide in Monster");
    }
}
```

```
class Witch extends Monster
{
    public Witch( String name ) {
        super(name);
    }
    public void overRide( ) {
        System.out.println("overRide in Witch");
    }
}
```

**Open
override.java**

method override

You cannot override the original method if it was defined as final.

```
public void final overRide( ) {  
    out.println("overRide in Monster");  
}
```

method override

```
class Monster
{
    private String myName;
    public Monster( String name ) {
        myName = name;
    }
    public final void overRide( ) {
        System.out.println("overRide in Monster");
    }
}
```

```
class Witch extends Monster
```

```
{
    public Witch( String name ) {
        super(name);
    }
    public final void overRide( ) {
        System.out.println("overRide in Witch");
    }
}
```

illegal – will not compile

**Open
final.java**

What is composition?

Composition

Composition is similar to inheritance, but is not inheritance. Composition occurs when one class contains an instance of another class.

***x has a Y* – X is composed of a Y**

Composition

```
public class Word implements Comparable  
{
```

```
    private String word;    //has a
```

```
    public Word(String w) { word = w; }  
    public int compareTo(Object obj)  
    {
```

```
        Word other = (Word)obj;
```

```
        if(word.length()>other.word.length())
```

```
            return 1;
```

```
        else if(word.length()<other.word.length())
```

```
            return -1;
```

```
        return 0;
```

```
    }
```

```
    public String toString() { return word; }  
}
```

Why can
you not
extend
String?

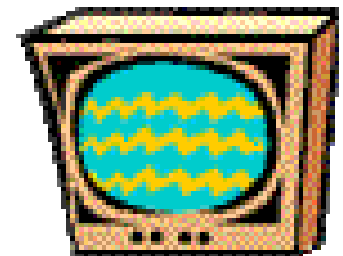
**Open
comppone.java**

What is static?

static

Static is a reserved word use to designate something that exists in a class.

Static variables and methods exist even if no object of that class has been instantiated.



static

Static means one!

All Objects will share the same static variables and methods.

static

```
class Monster
```

```
{
```

```
    private String myName;
```

```
    private static int count = 0;
```

all Monster share count

```
    public Monster() {
```

```
        myName = "";
```

```
        count++;
```

```
    }
```

```
    public Monster( String name ) {
```

```
        myName = name;
```

```
        count++;
```

```
    }
```

```
}
```

Open static.java

KeyListener Interface

KeyListener

abstract methods

Name	Use
keyPressed(e)	called when a key is pressed
keyReleased(e)	called when a key is released
keyTyped(e)	called when a key has been typed

```
import java.awt.event.KeyListener;
```


**Continue work
on Lab 02**