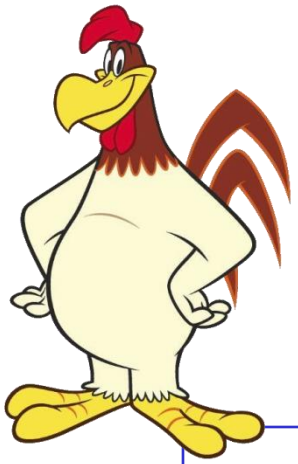
Inheritance

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 only 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();
System.out.println(one);
one = new B();
System.out.println(one);
```

OUTPUT

```
8
8
```

inheritance example

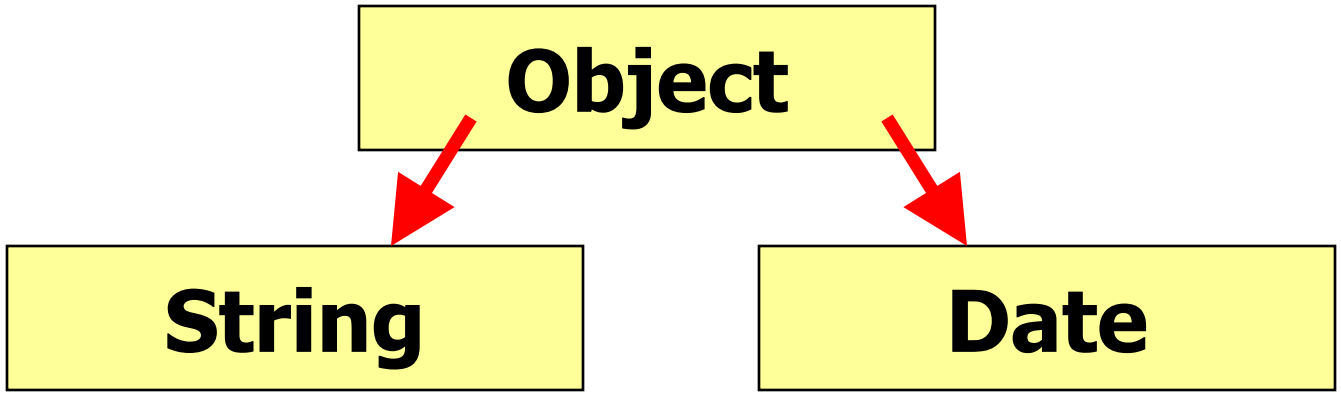
```
class A
{
   private int x;
   public A() { x =3;}
   public void setX(int val)
   {
      x=val;
   }
   public int getX(){ return x; }
}

class B extends A{}

//test code in the main method
B one = new B();
System.out.println(one.getX());
one.setX(2);
System.out.println(one.getX());
```

OUTPUT

3
2

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
```

Overriding methods like toString() and equals() is considered good programming practice.

What's on the inside?

A String Object

Object methods
Object variables
String methods
String variables

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
{

}
```

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 level access.

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 then be written using the super class methods with no real concern for the implementation details in the super class methods.

this

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

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

this.x = 1524; Sets the value of x for this object

this( ); calls a constructor of this class

this

```
class Monster
{
    private String myName;
    public Monster()
    {
        this("Monster");
    }
    public Monster( String name )
    {
        myName = name;
    }
    public String toString()
    {
        return myName + "\n";
    }
}
```

calls Monster(name)

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

super this

```
class Skeleton extends Monster
{
    private double speed;

    public Skeleton( )
    {
        speed=100;
    }

    public Skeleton( String name, double speed )
    {
        super(name);
        this.speed=speed;
    }

    public String toString( )
    {
        return super.toString() + " " + speed;
    }
}
```

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

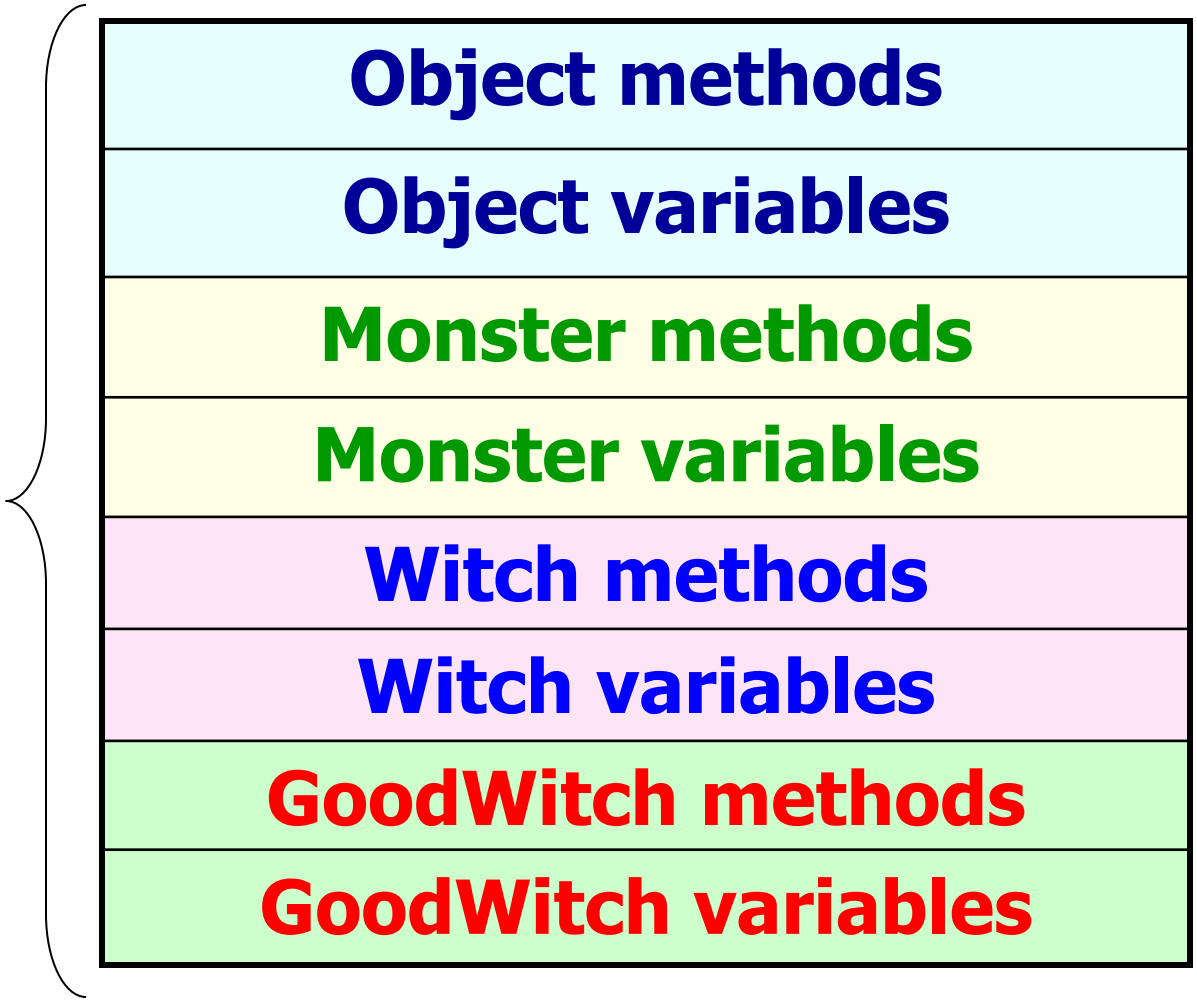
super – refers to the parent – calls the constructor.

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



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

In Java, a parent can always refer to a child. Object is the parent of every other class in Java, so it is perfectly okay for Object to refer to a String.

Polymorphism

```
Object x = "compsci";
System.out.println(x.toString());
```

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

OUTPUT

compsci

Because Object has a `toString()`, it is okay to call the `toString()` method on x. Java will dynamically call the String `toString()` at run time. At runtime, Java will call the `toString()` on whatever type of Object x refers to.

Polymorphism

```
Object x = "compsci";

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

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

OUTPUT

syntax error

Java sees x as an Object reference at compile time and checks to see if Object has a `length()` method. It finds that Object does not have a a `length()` method; thus, Java reports a compile error.

Polymorphism

```
Object x = "compsci";

System.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?

A child can NEVER refer to a parent.
Inheritance works one way… not the other.

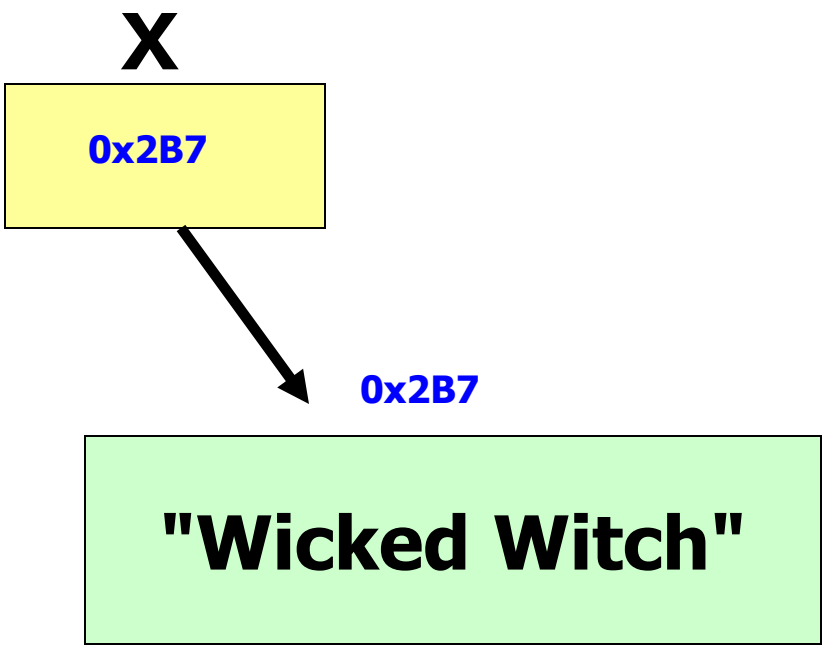
Polymorphism

```
Monster x = new Witch();
Monster y = new Ghost();
System.out.println(x);
System.out.println(y);
```

Is this okay or not okay?

A parent can always refer to a child. The more general reference goes on the left of the equals sign and the equal or more specific object instantiation goes on the right hand side.

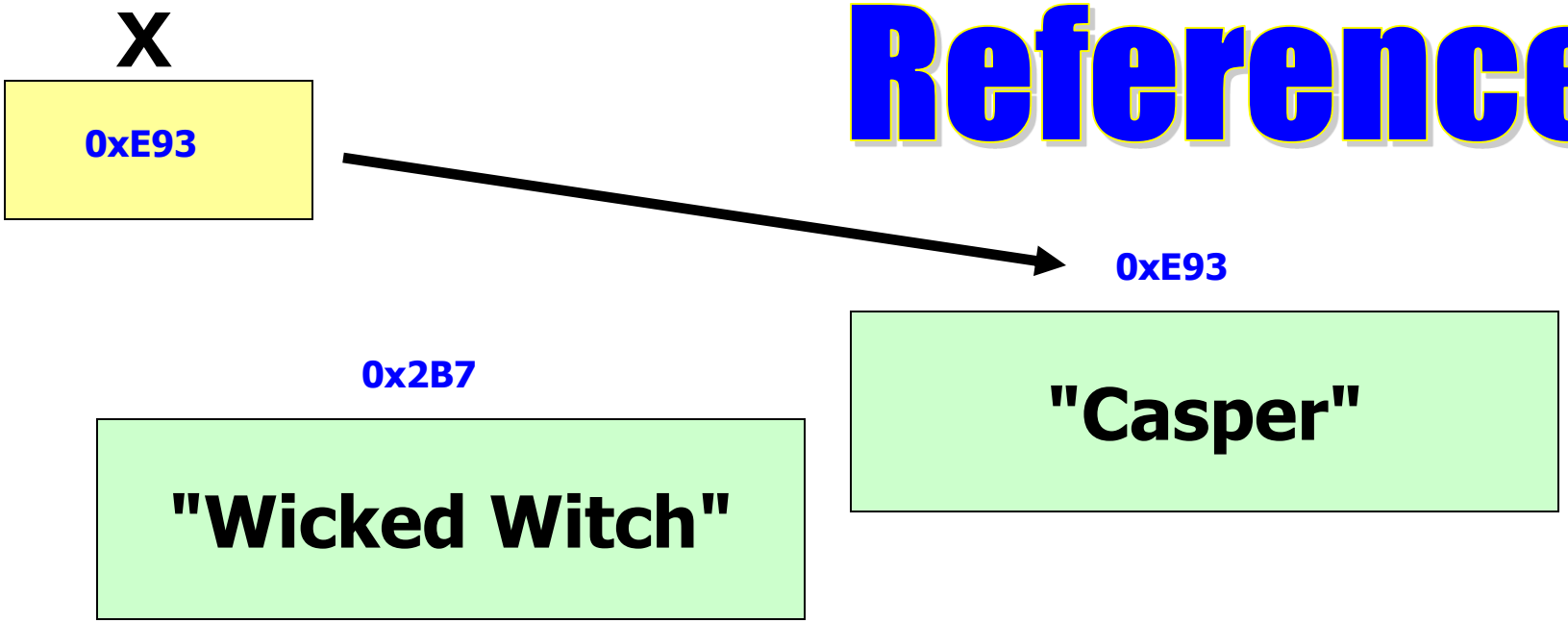
References



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!

method override

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

You can override a parent method 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");
     }
}
```

Final

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

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

Final

```
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

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?

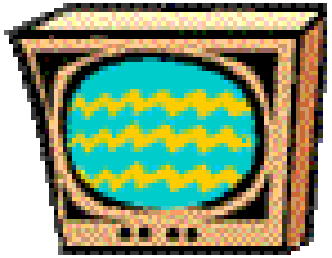
Because it is a final class!

What is static?

static

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

Static variables and methods are bound to a class not an object instantiation.



static

Static means there is just one copy of it!

All Objects (if there are any)
will share the same static variables and methods.

static

```
class Monster
{
    private String myName;
    private static int count = 0;

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

all Monster objects share count

```java
class Static
{
  public static void main ( String[] args )
  {
     System.out.println("\nMonster count = " + Monster.getCount() + "\n\n");

     Ghost casper = new Ghost("Casper");
     System.out.println(casper);
     System.out.println("Monster count = " + casper.getCount() + "\n");

     Witch witch= new Witch("Harriet");
     System.out.println(witch);
     System.out.println("Monster count = " + witch.getCount() + "\n");

     Ghost gone = new Ghost();
     Ghost gtwo = new Ghost();
     Ghost gthree = new Ghost();
     Witch west = new Witch("Wicked Witch Of the West");
     System.out.println(west);
     System.out.println("Monster count = " + west.getCount() + "\n");
  }
}
```