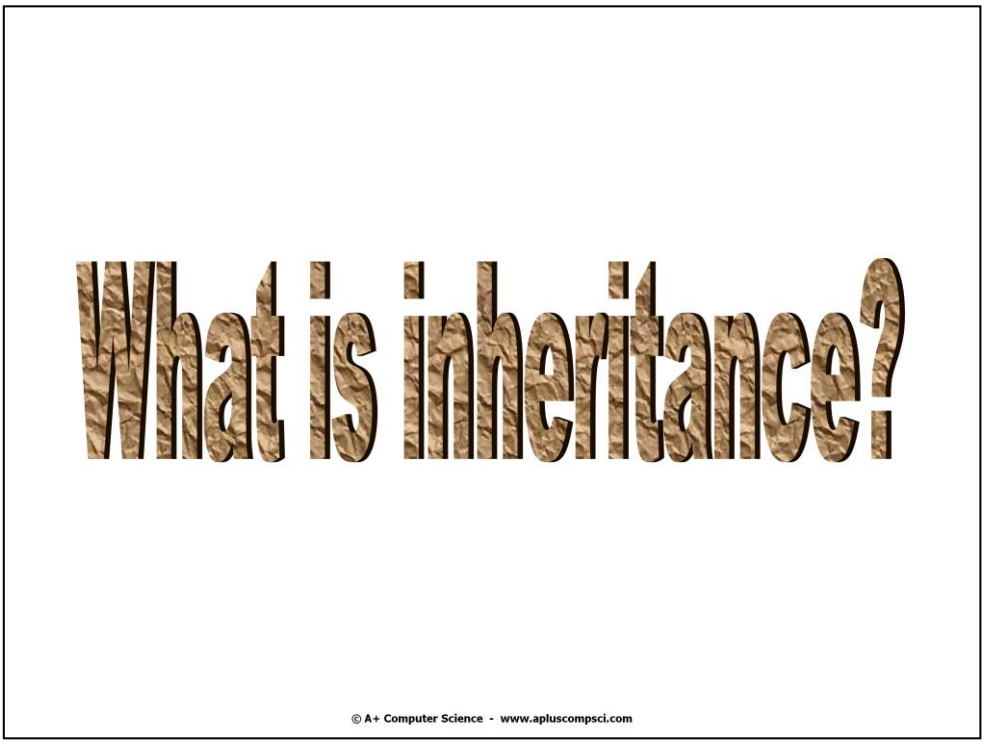
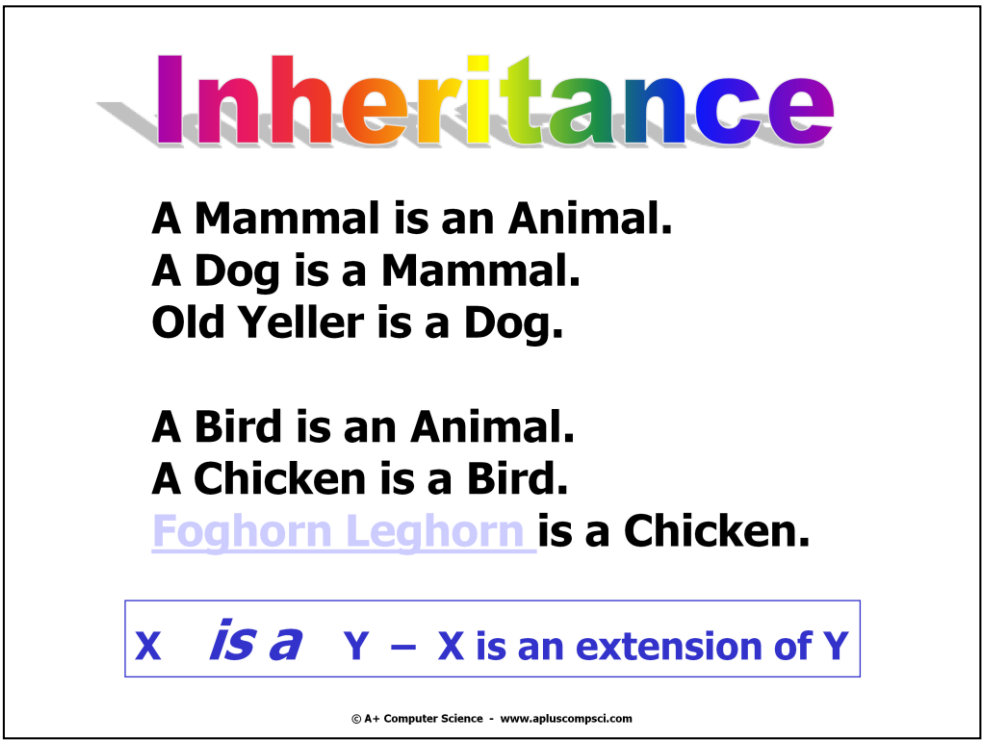
Inheritance

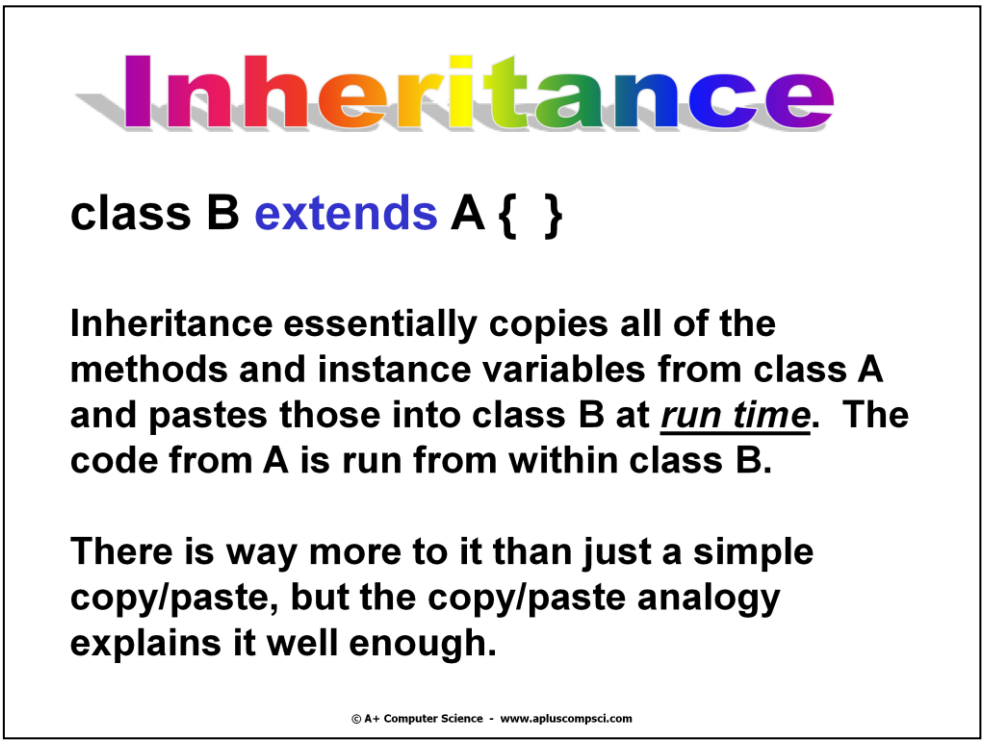

What is inheritance?
© A+ Computer Science - www.apluscompsci.com



A Mammal is an Animal. A Dog is a type of Mammal. Old Yeller is a specific Dog. Old Yeller is an Animal and a Mammal.

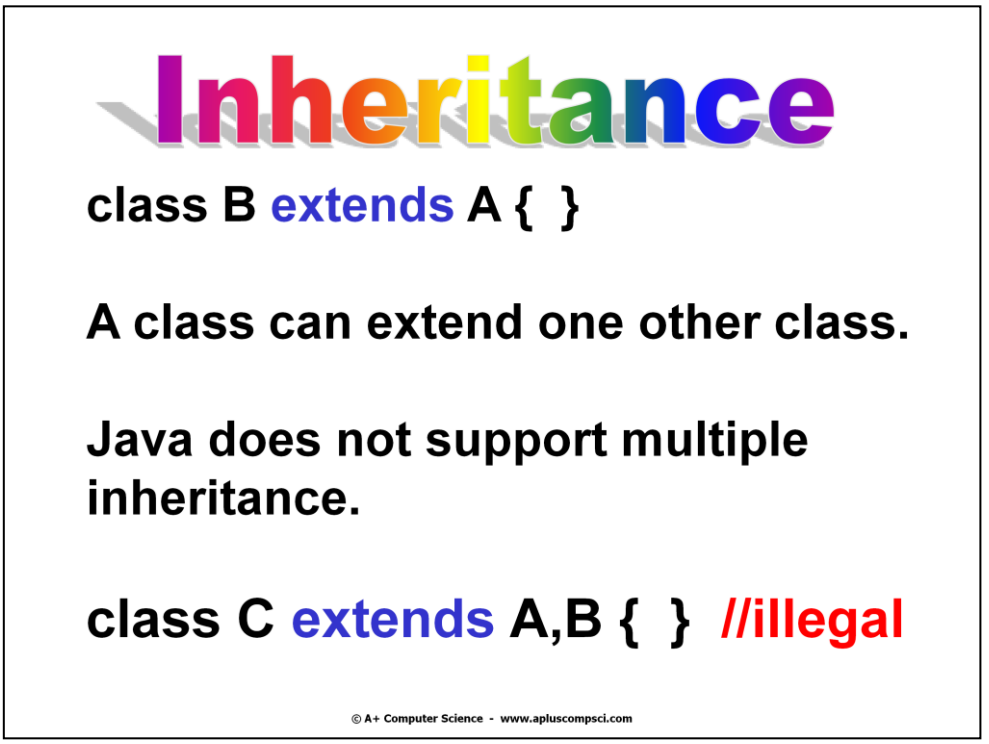
A Bird is an Animal. A Chicken is a type of Mammal. Foghorn Leghorn is a specific Chicken even though he denies it. Foghorn Leghorn is an Animal and a Mammal.

Inheritance deals with relationships and building new things using old things. A Mammal is an extension of an Animal; thus, a Mammal starts off with all of same stuff as an Animal. A Mammal may have more stuff than an Animal, but it will have at the very least the same stuff as an Animal.



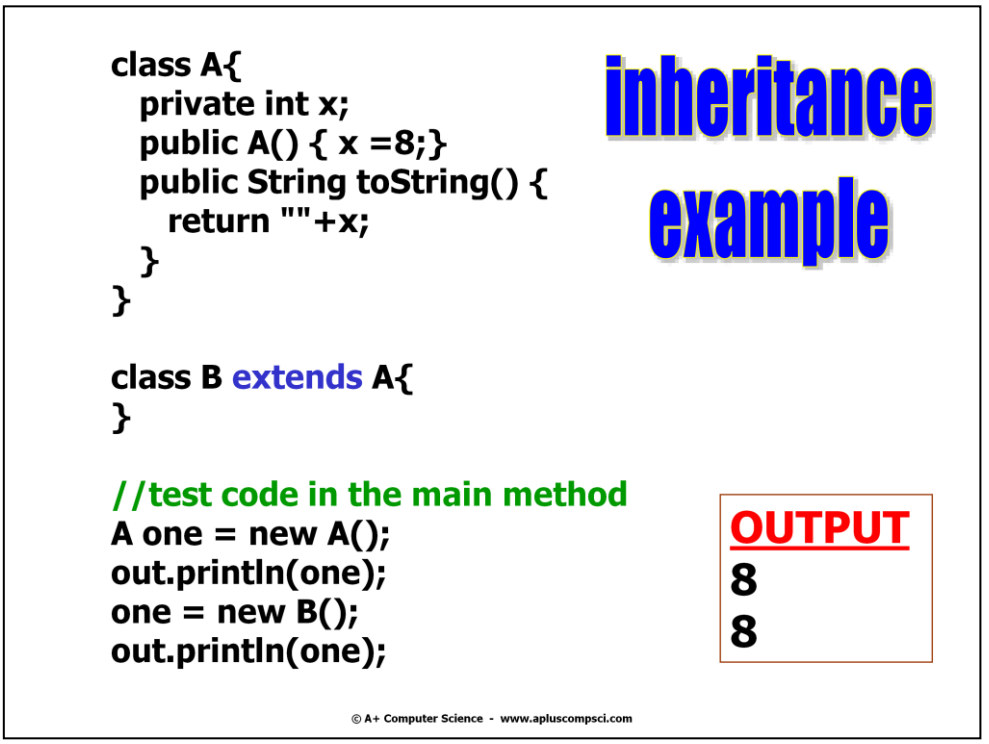
In Java, inheritance is used to create relationships. If B is an A, then B is related to A in that B is a child of A. B starts out with all of the same stuff as A. B may have more stuff than A, but B will have at least the same stuff as A.

If class A has 3 methods and two variables, then class B will start out with at least 3 methods and two variables.



Java allows one class to extend one other class. Java does not allow one class to extend more than one other class.

A class may implement as many interfaces as required, but a class may only extend one other class.

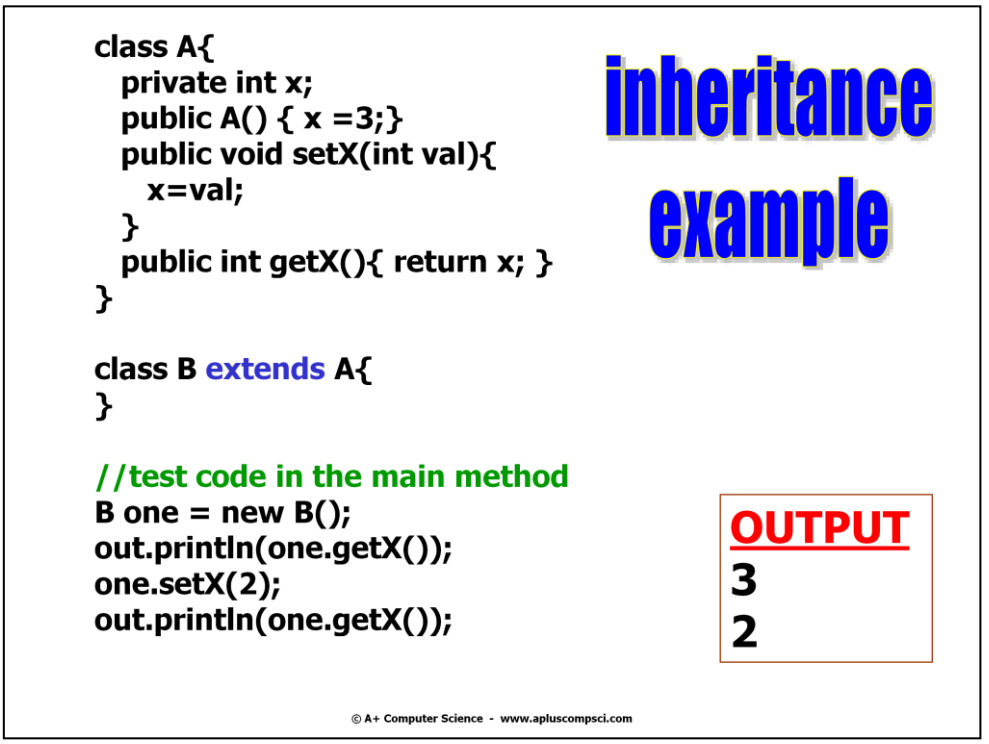


Class B extends A which means that B is a sub class of A. When looking at B, there are no visible methods or visible data of any kind.

Because B extends A, B will contain all methods and data from A. Some parts of A will not be accessible in B and some of the parts will be accessible. All parts of A with public access are fully accessible in B. All parts with private access are not accessible in B.

When `one` is printed in the main the first time, `one` is referring to an A object. Java calls the A `toString()` and prints out an 8.

When `one` is printed the second time, `one` is referring to a B object. B does not have its own `toString()` method so Java uses the only `toString()` present which is the `toString()` method that B inherited from A.



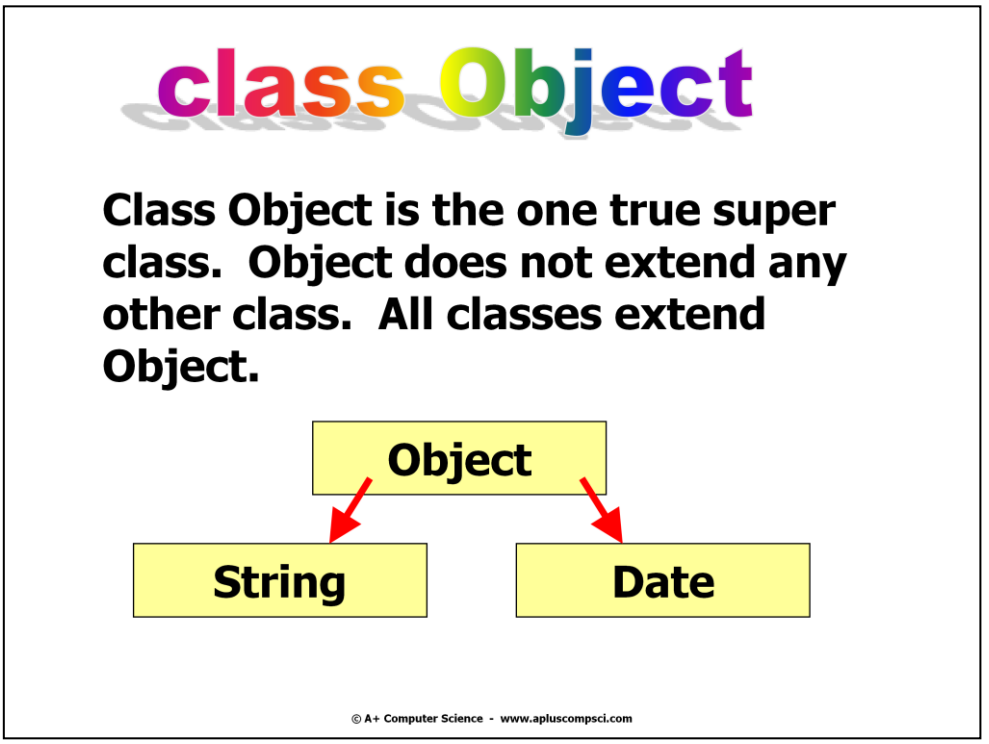
Class B extends A which means that B is a sub class of A. When looking at B, there are no visible methods or visible data of any kind. Because B extends A, B will contain all methods and data from A.

one is defined as a B reference. one is referred to a new B Object.

The first println prints out 3 because the default A constructor sets x to 3.

The second println prints out 2 because the setX method set x to 2.

Open inheritone.java



Class Object is the parent of every other class in Java. All classes in Java start out with all of the same data / instance variables and methods as class Object.

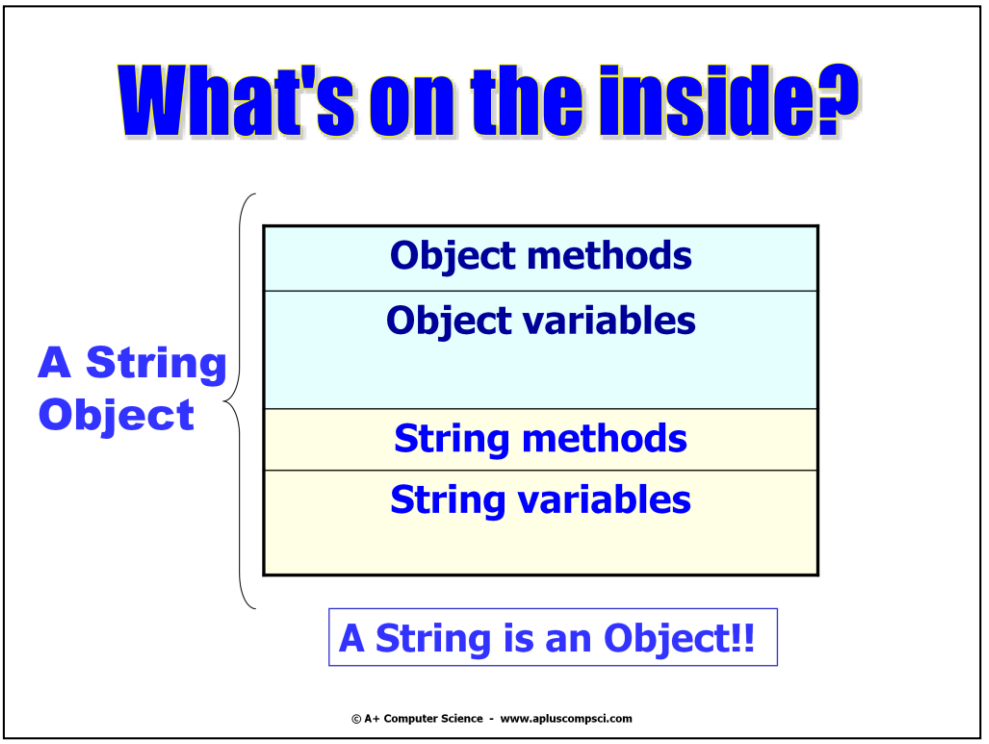
class Object

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

The most common Object methods used are `equals()`, `toString()`, `clone()`, and `hashCode()`. There are many more as well, but these are the more common ones.

It is considered a very good practice to override the Objects method in a new class. Writing specific toString() and equals() methods for instance is considered a good habit.



Because String extends Object, String will contain all methods and instance variables / data from Object as well as all methods and instance variables / data written in class String.

String *is an* Object.

```
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
{

}
```

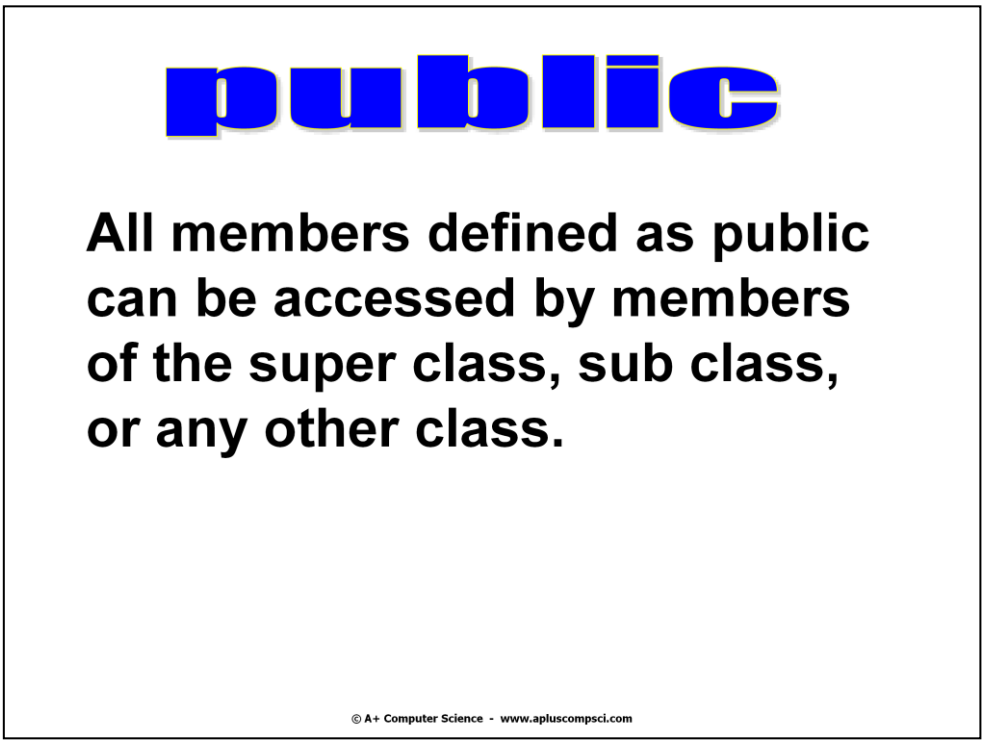
inheritance example two

Class Witch is a child of Monster. Witch does not have any visible methods or data.

Witch will contain all of the same methods and instance variables / data as class Monster as well as all of the same methods and instance variables / data as class Object.

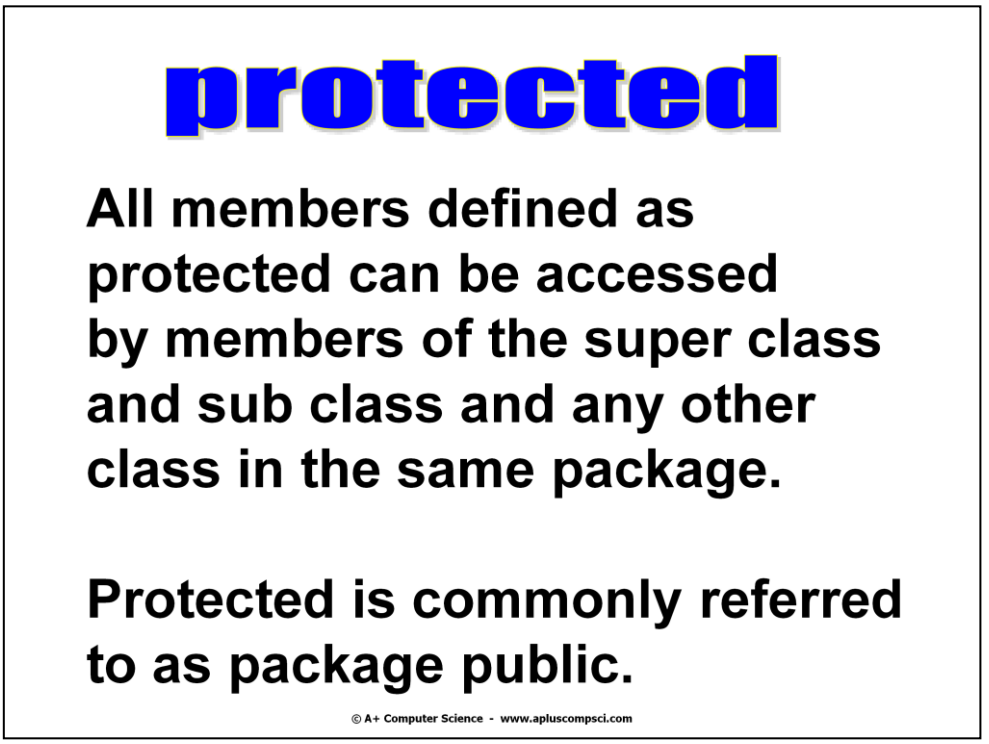
Open inherittwo.java


Public
Protected
Private
© A+ Computer Science - www.apluscompsci.com



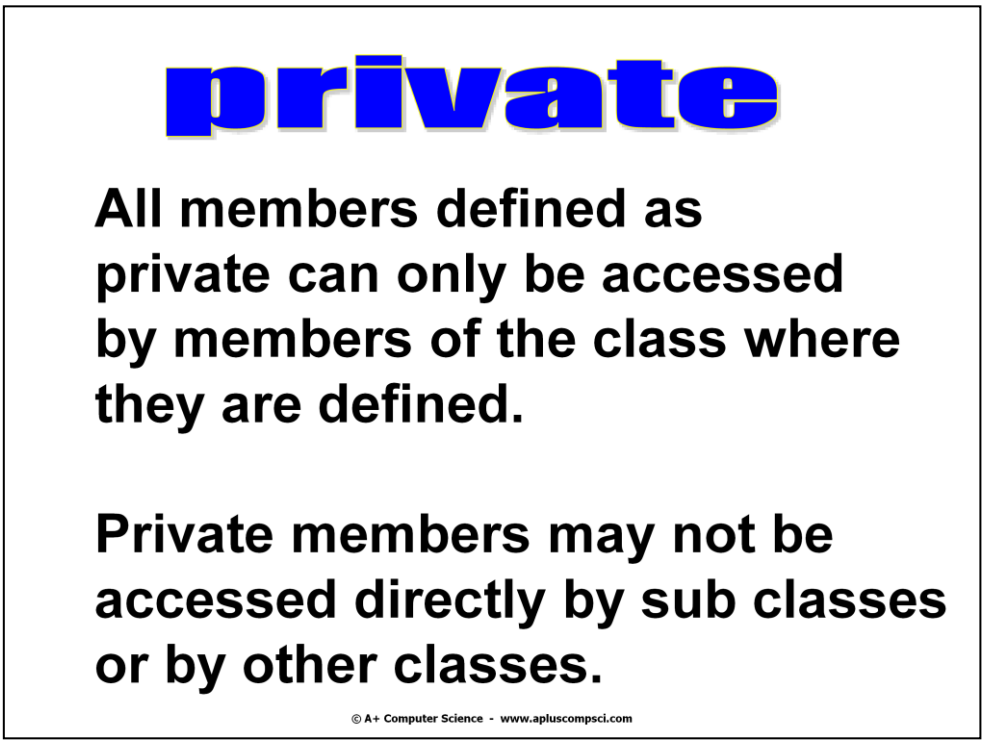
When methods or data are labeled public, they can be accessed in the super class, sub class, or any other class.

When a sub class inherits public parts, those parts can be accessed directly by the sub class.



When methods or data are labeled protected, they can be accessed in the super class and sub class only.

When a sub class inherits protected parts, those parts can be accessed directly by the sub class.



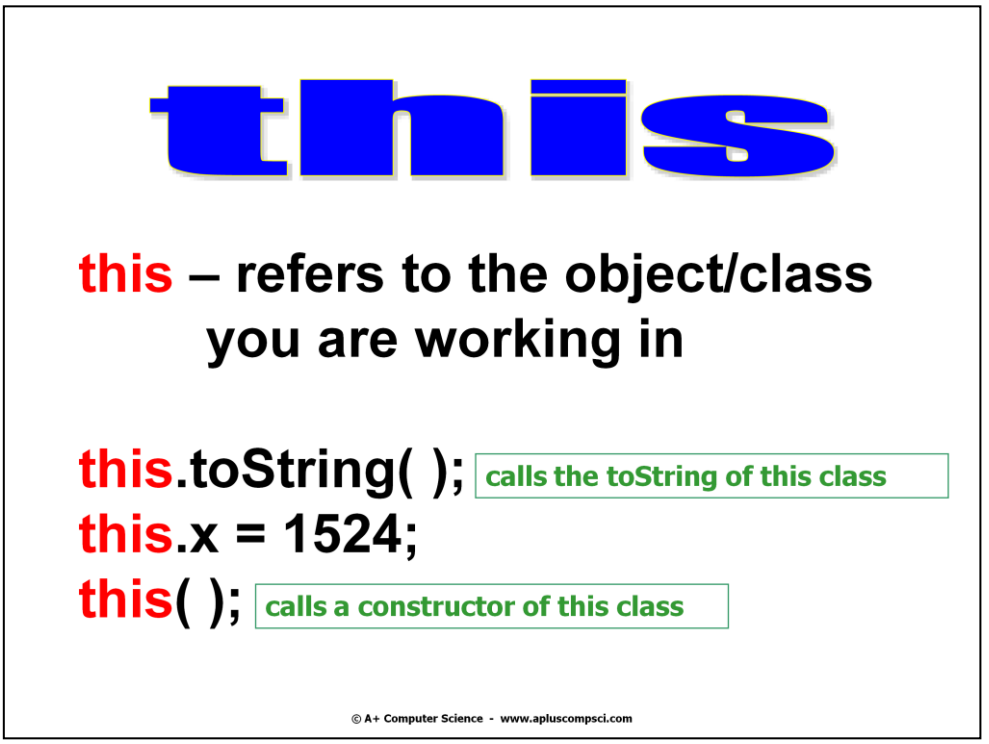
When methods or data are labeled private, they can be accessed in the super class.

When a sub class inherits private parts, those parts cannot be accessed directly by the sub class or any other class.

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



Can you see this?

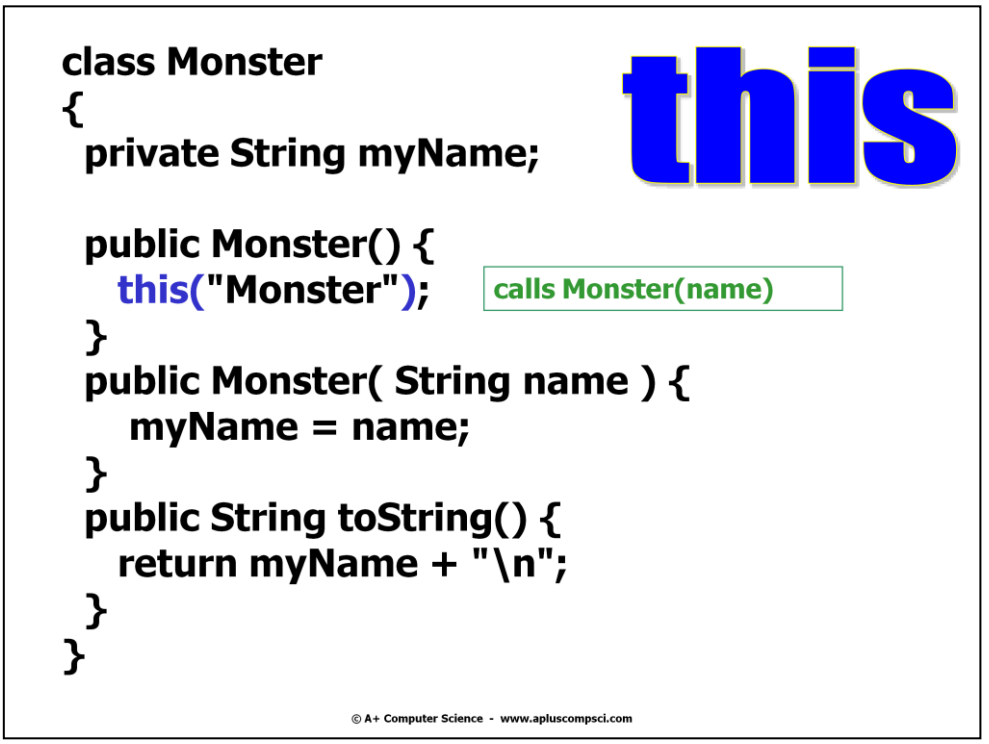
Are you getting this?

Does this make sense?

Do I need to repeat this?

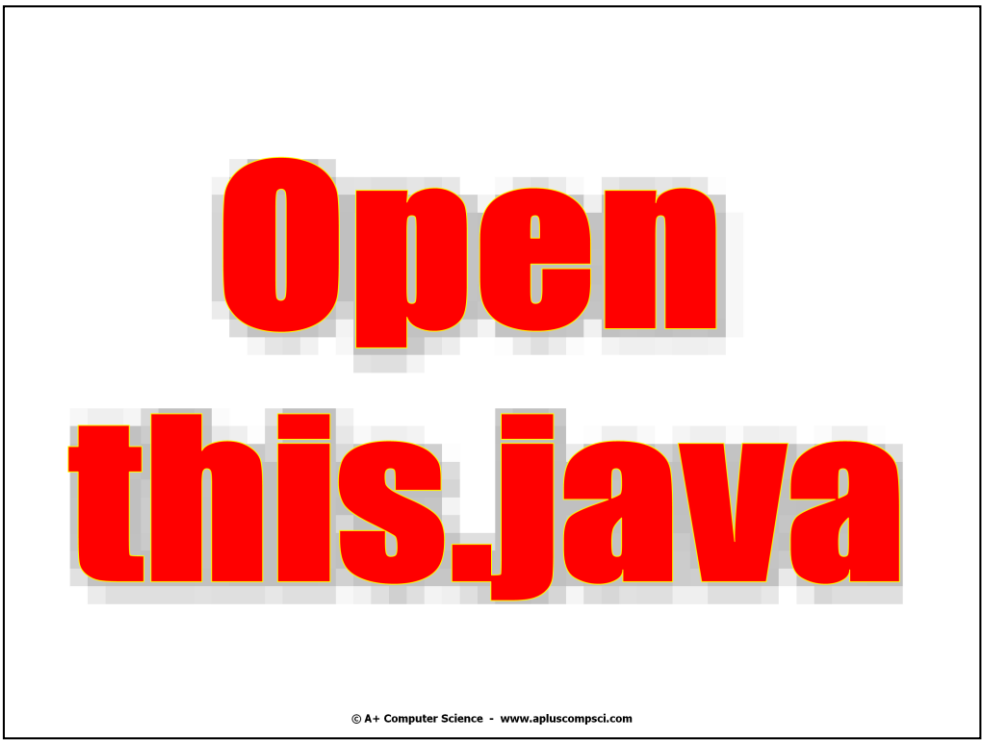
This is very useful.

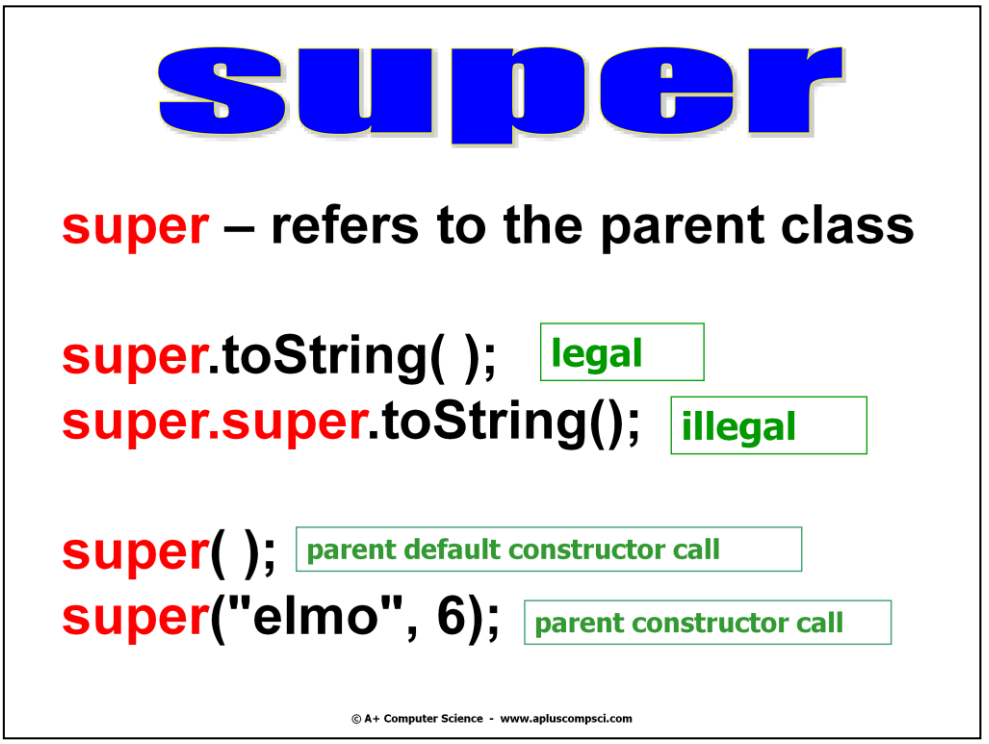
`this` is used to refer to the object in which the work is being done. `this` can be used when you need to access the entire object as a whole.



This is very useful when calling constructors of the same class.

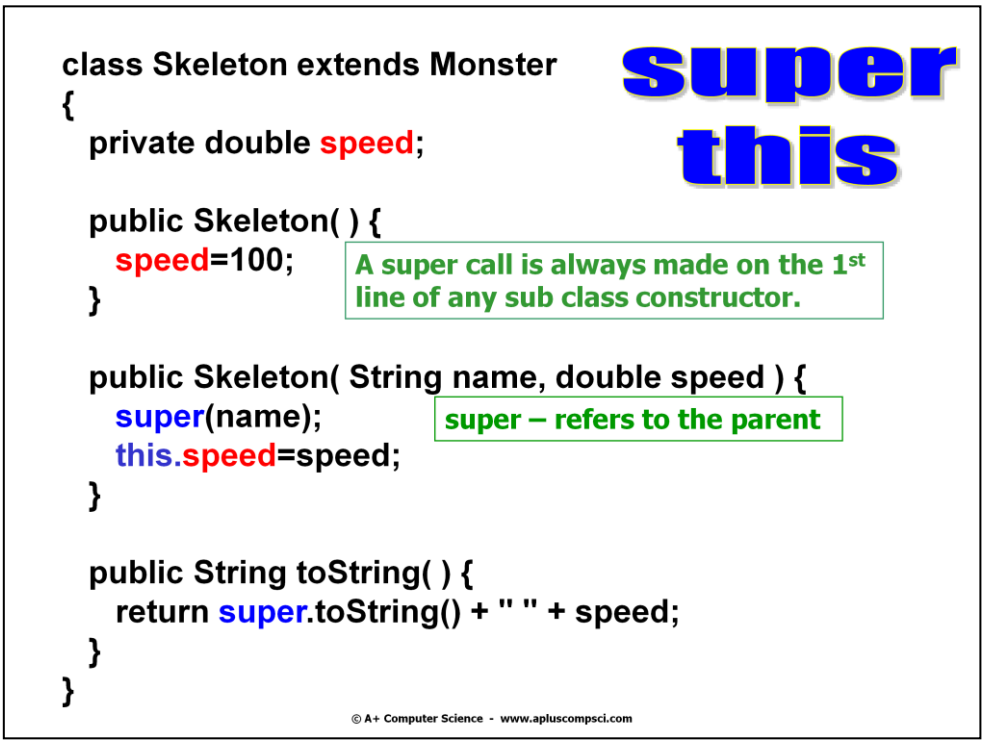
When working in Monster, you could not write `Monster("fred")` to call the `Monster(String name)` constructor. You would have to use the `this("fred")` call instead.


Open
this.java
© A+ Computer Science - www.apluscompsci.com



Super is typically used to call parent class constructors and to call parent methods that have been overridden in the sub class.

Super is necessary to distinguish between methods when the sub class and super class have a method with the same signature.



In this example, Skeleton has its own data, but needs to make sure the Monster data is setup properly. In each of the Skeleton constructors, a call is made to the appropriate `super()` constructor to ensure that Monster has been instantiated properly. If a call is not made to a `super()` constructor, Java will automatically call the default `super()` constructor on the first line of every sub class constructor.

In the `toString()` method of Skeleton, the `super().toString()` is called to retrieve the super class data.

If the `super.` was not placed in front of the toString() call, a recursive process would begin.

Open superthis.java

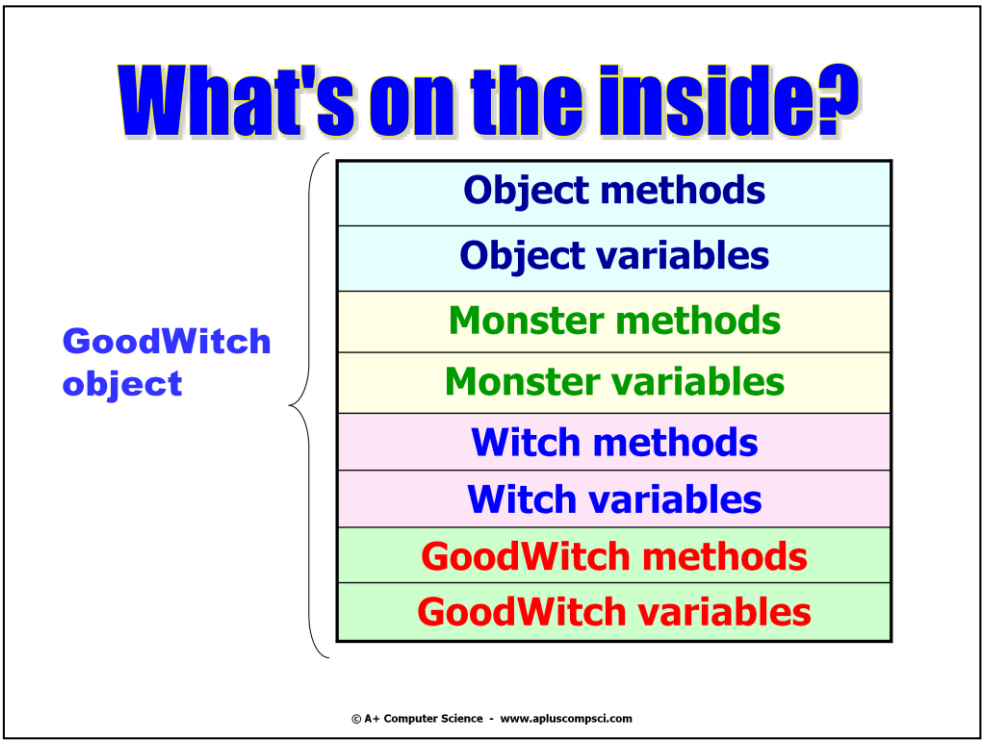
Start work on Lab 20

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( )   }
}
```


In the example, GoodWitch extends Witch. The GoodWitch constructor will call the Witch() constructor automatically via a `super()` call. If the Witch() constructor did not exist, the code would not compile.

If a particular `super()` constructor should be called, that `super()` call must be placed on the first line of the sub class method or the default `super()` will be called.



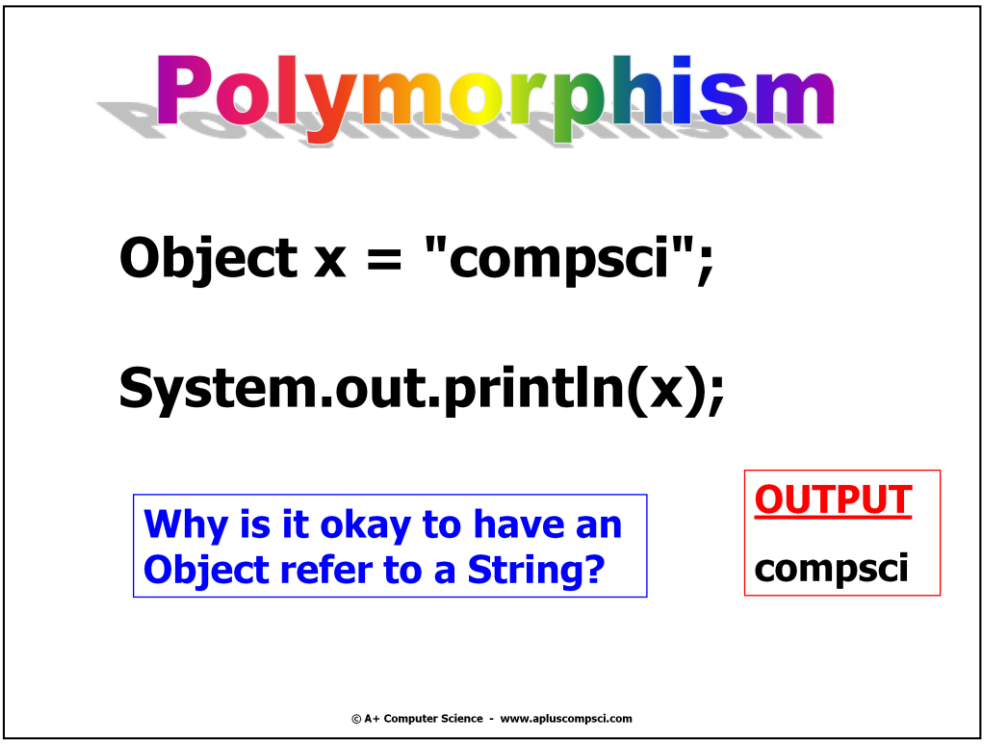
GoodWitch has a lot of stuff inside. GoodWitch gets all of the stuff from Witch, Monster, and Object. That is a lot of data and lots of methods.

Open whatsontheinside.java

Create 2 new monsters from the Monster class.

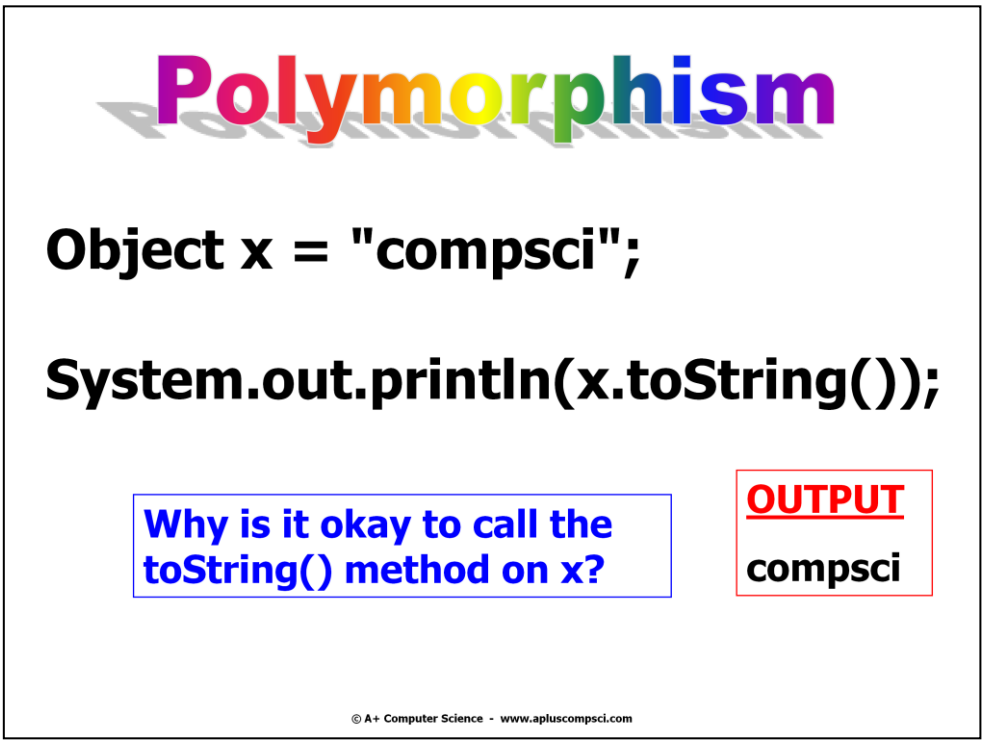
Polymorphism

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



In Java, a parent can always refer to a child. A child can never refer to a parent.

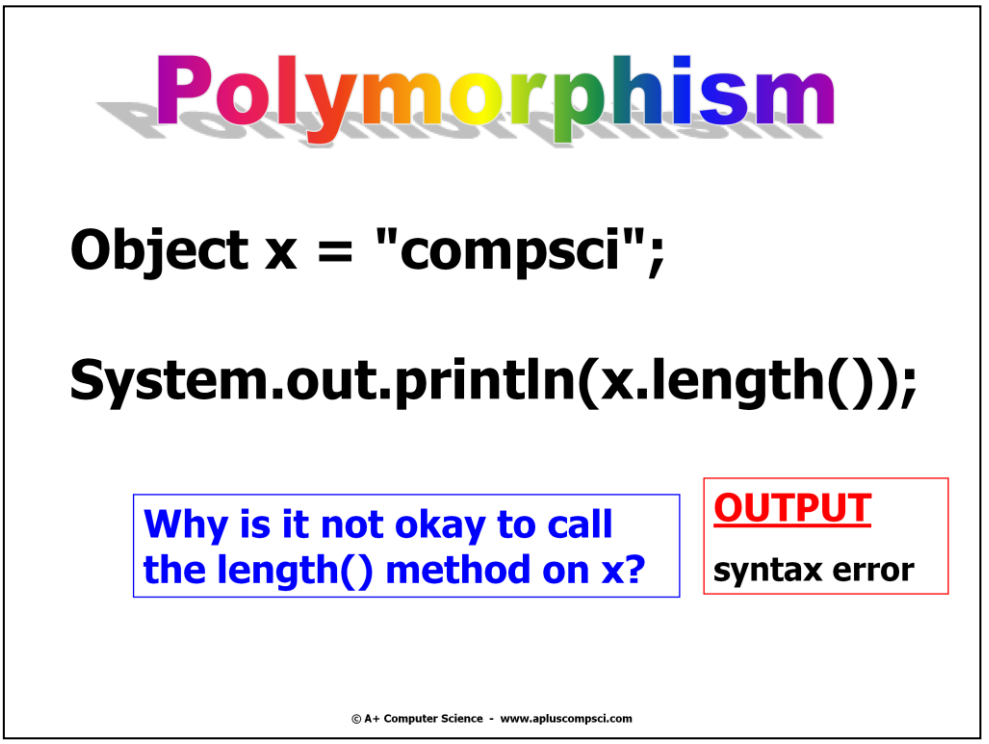
As Object is the parent of every other class in Java, it is perfectly okay for Object to refer to a String.



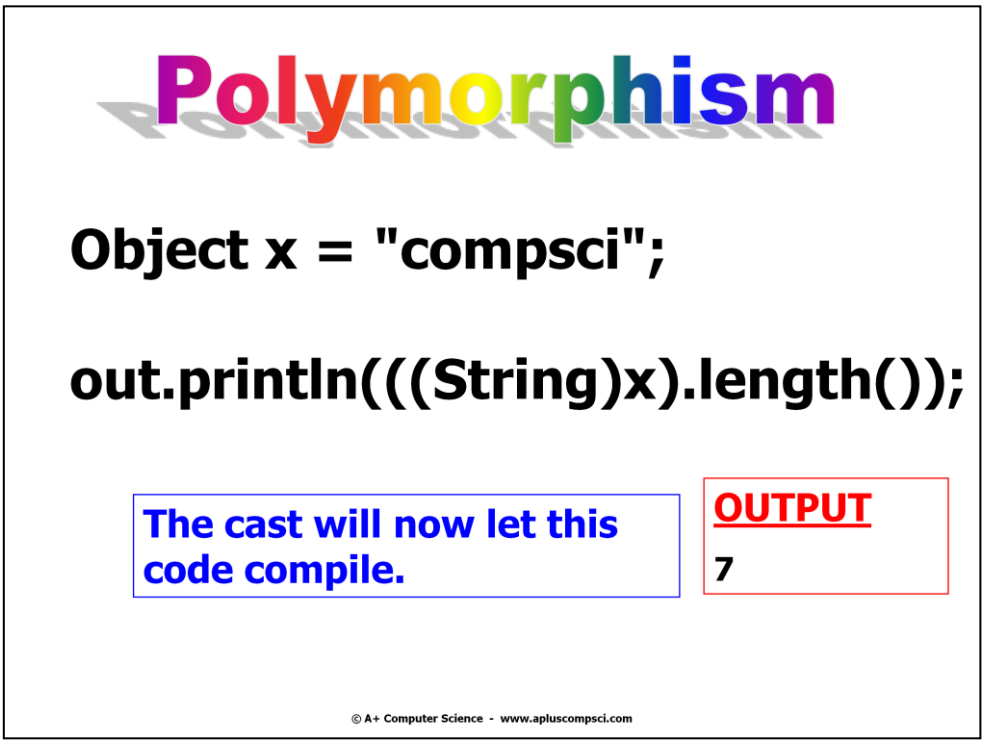
In Java, a parent can always refer to a child. A child can never refer to a parent.

As Object is the parent of every other class in Java, it is perfectly okay for Object to refer to a String.

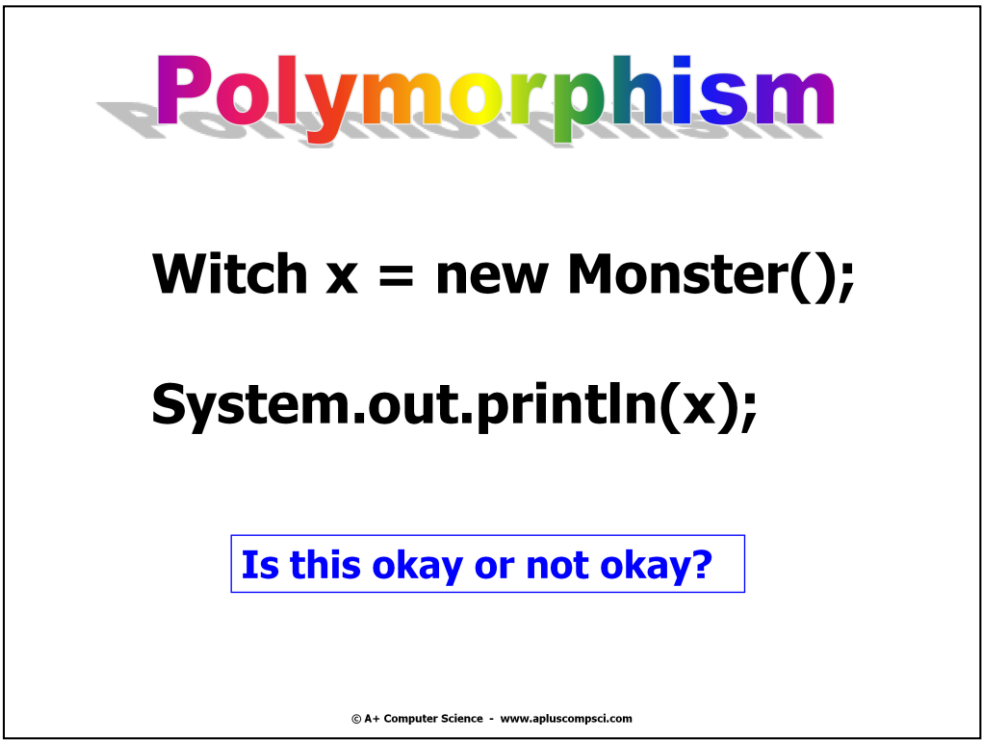
Because Object has a `toString()`, it is okay to call the `toString()` method on x. Java will dynamically call the String `toString()` at run time. This demonstrates polymorphic behavior about as clearly as you can. At compile time, Java checks to see that x has a toString(). At runtime, Java will call the `toString()` on whatever type of Object x refers to.



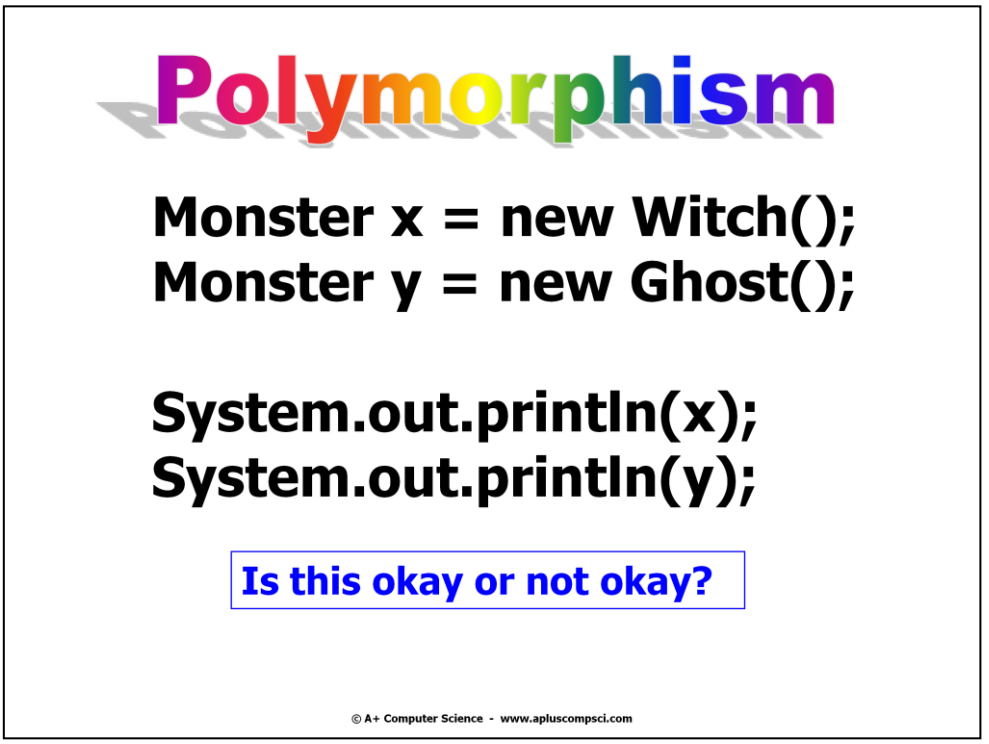
Because x is an Object reference and Object does not have a length() method, Java will not be able to compile this code. It is quite obvious to us that x is referring to a String Object and that String has a `length()` method, but Java does not have the same vision. Java sees x as an Object reference and 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.



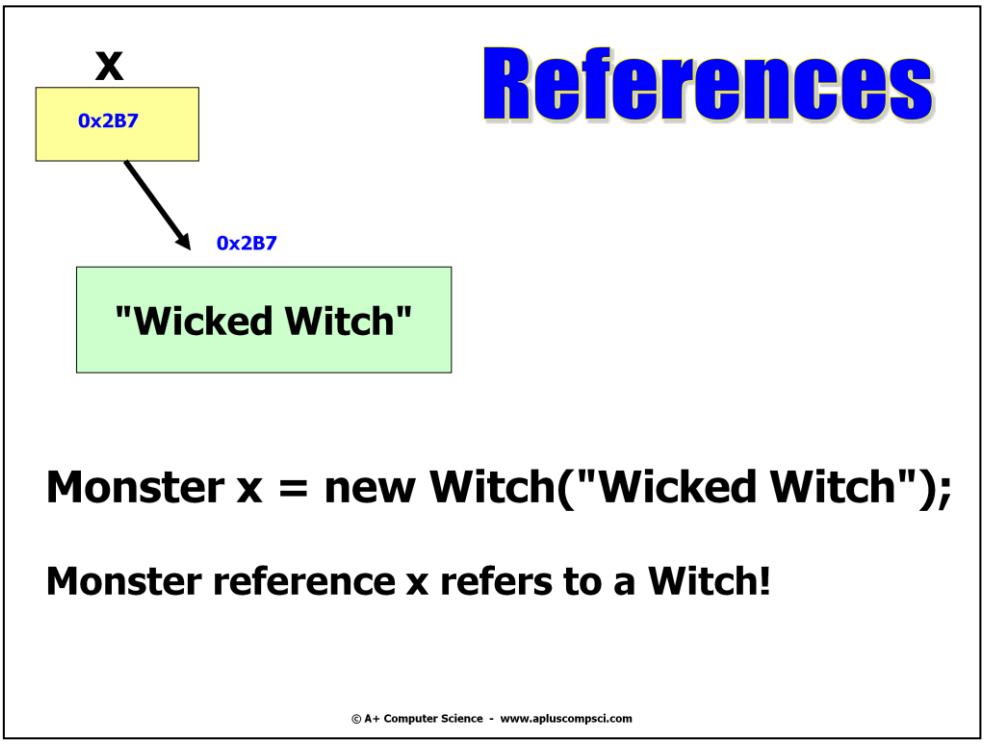
With the addition of the (cast), Java now realizes that x is really referring to a String and the compile error is removed.



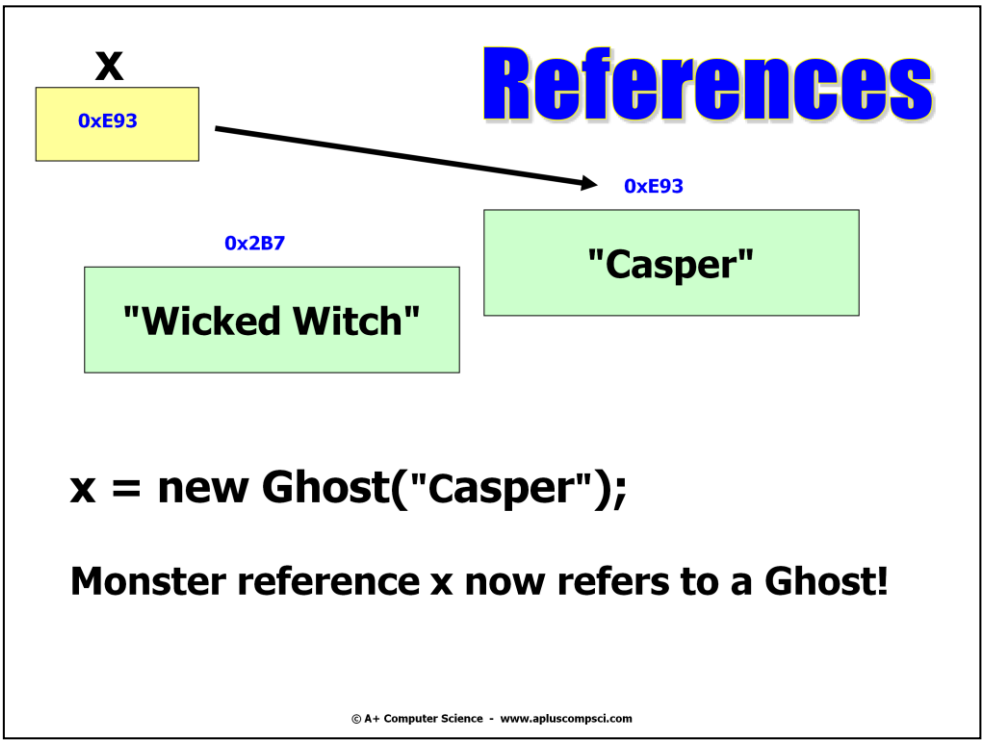
A child can never refer to a parent.



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

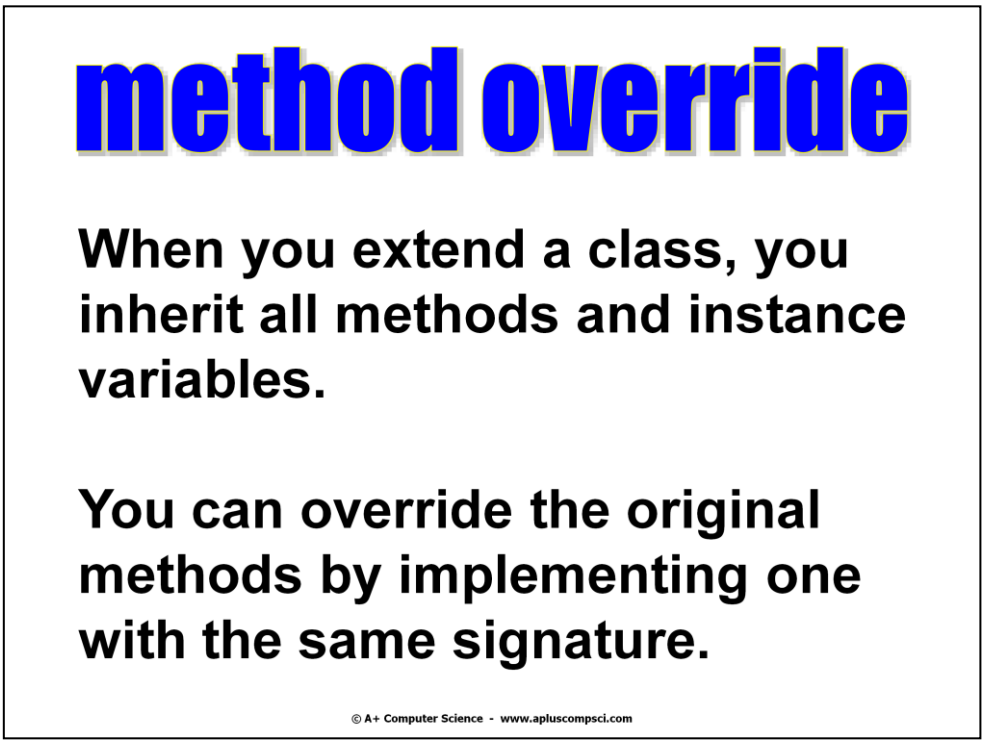


X is referring to a Witch.



X is now referring to a Ghost. The Witch object is now in limbo.

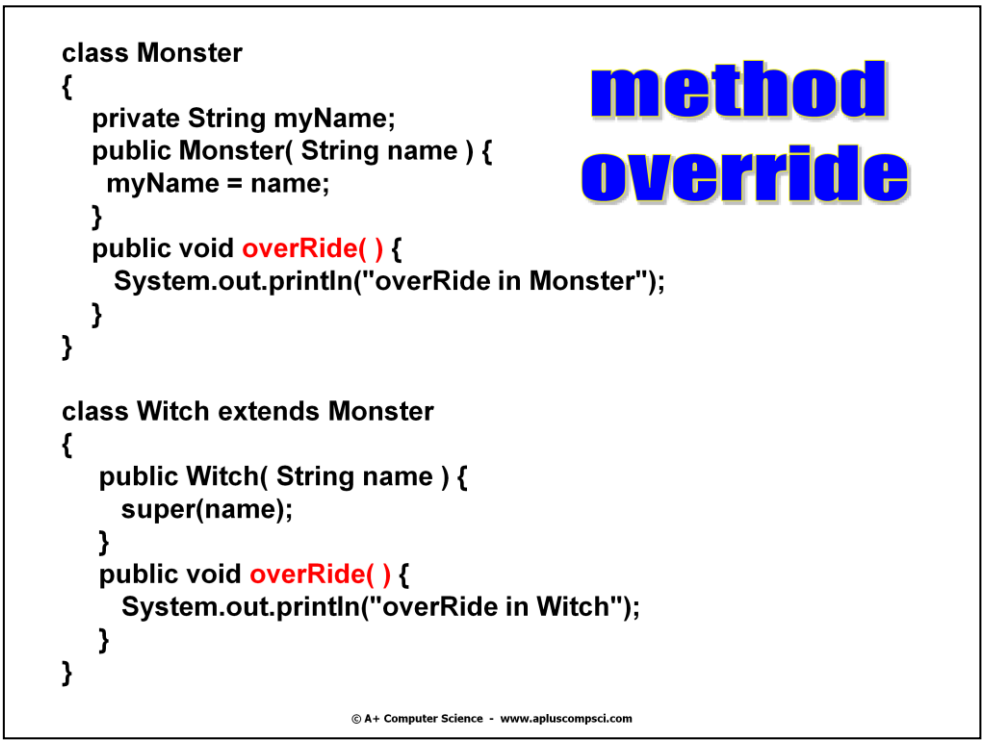
Open poly.java



When extending a super class, the sub class gets all of the data and methods from the super class.

There are times when the methods inherited need to be rewritten to be more specific. Overriding a method occurs when a new method is created in the sub class with the exact same signature as the super class method. The super class method still exists and could be called using super.

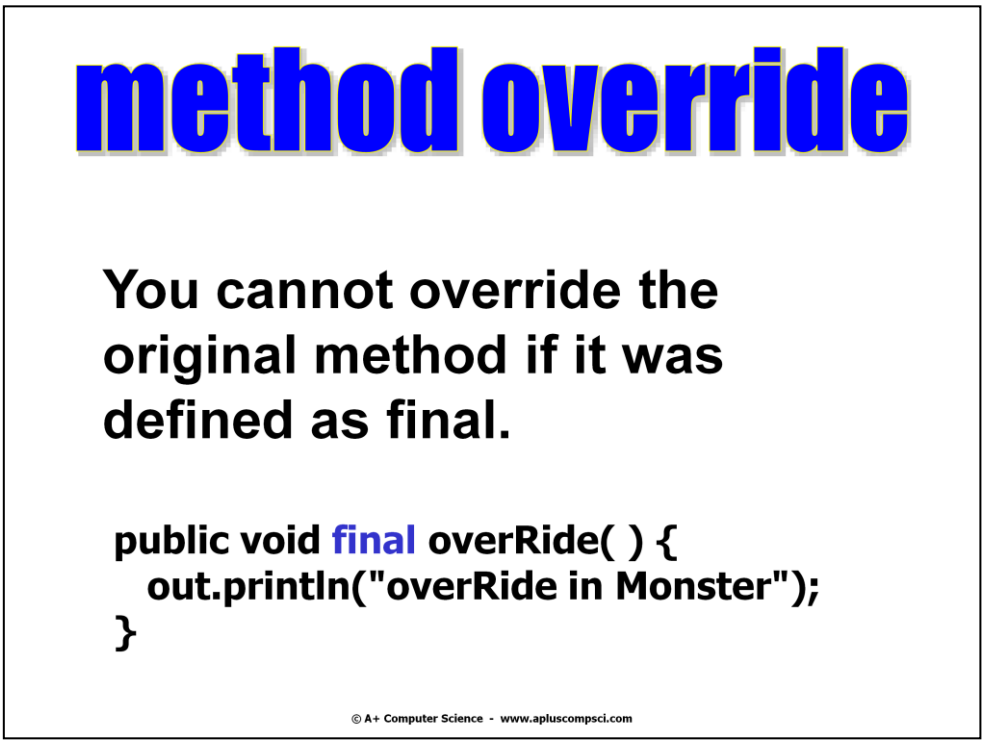
As everything in Java is an Object, method overriding occurs quite frequently. Anytime a toString() is written for a class, the Object toString() is being overridden.



Class Witch gets all data and methods from Monster. One of the methods received from Monster by Witch is override.

The override method is rewritten in class Witch with the same signature as the original Monster method. The Monster override method has been overridden in class Witch. In order to use the Monster override, a `super.overRide()` call would have to be made.

Open override.java



Final is used to indicate something that should not be changed.
If a method is labeled final, that method cannot be overridden.
If a class is labeled final, that class cannot be extended.

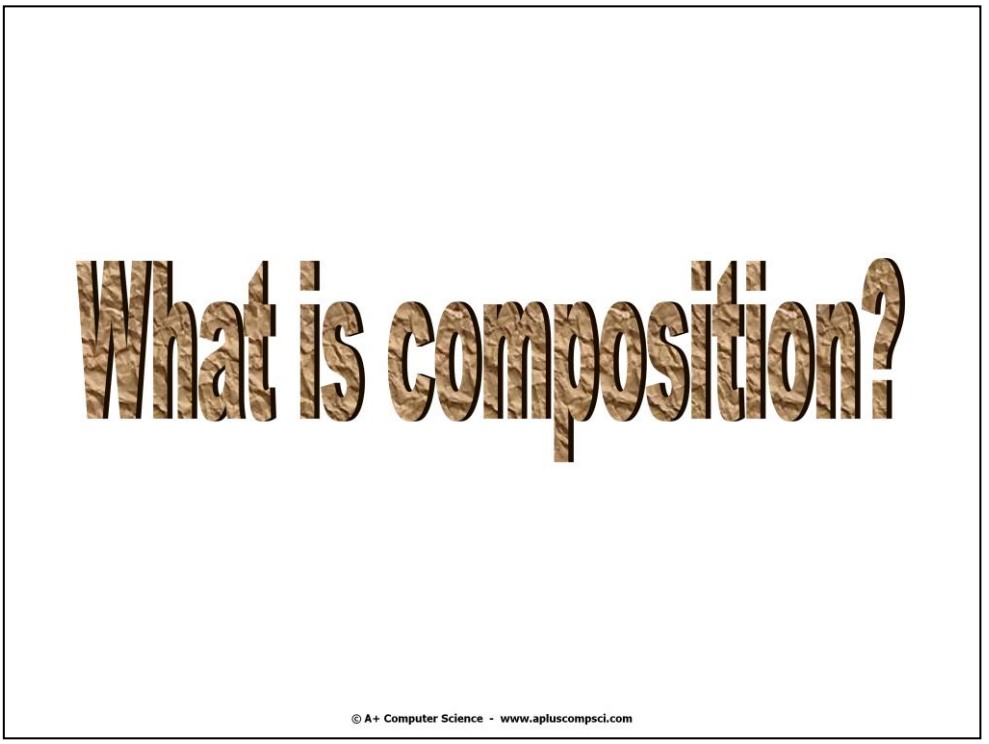
```
class Monster
{
   private String myName;
   public Monster( String name ) {
    myName = name;
   }
   public final void overRide( ) {
     System.out.println("overRide in Monster");
   }
}

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

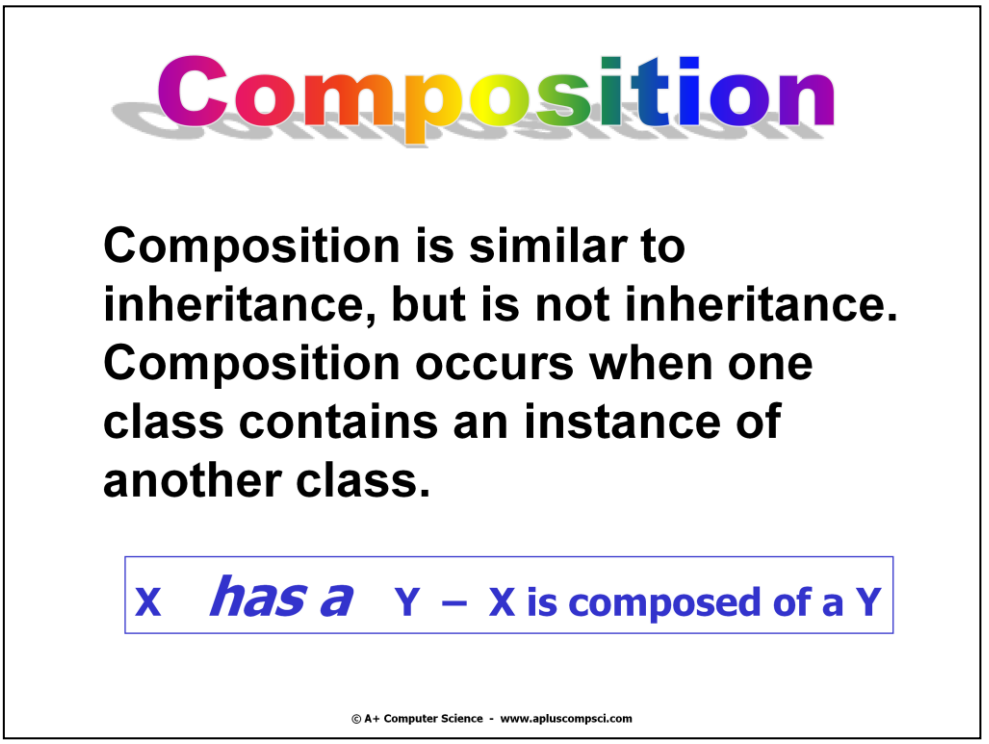
method override

illegal – will not compile

Because the `overRide()` in Monster is final, the `overRide()` method cannot be overridden in Witch. Monster could contain an `overRide()`, but not Witch.

Open final.java


What is composition?
© A+ Computer Science - www.apluscompsci.com



Composition is very useful and similar to inheritance. Composition occurs when a class contains an instance variable of another class type. The new class is composed of parts of which are other classes.

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?

Word is not a String, but it does contain a reference to a String. Word has a String, but is not a String.

This would be a great place for inheritance in that it makes a lot of sense for Word to simply extend String, but String is a final class and it cannot be extended.

Open compone.java

What is static?

static

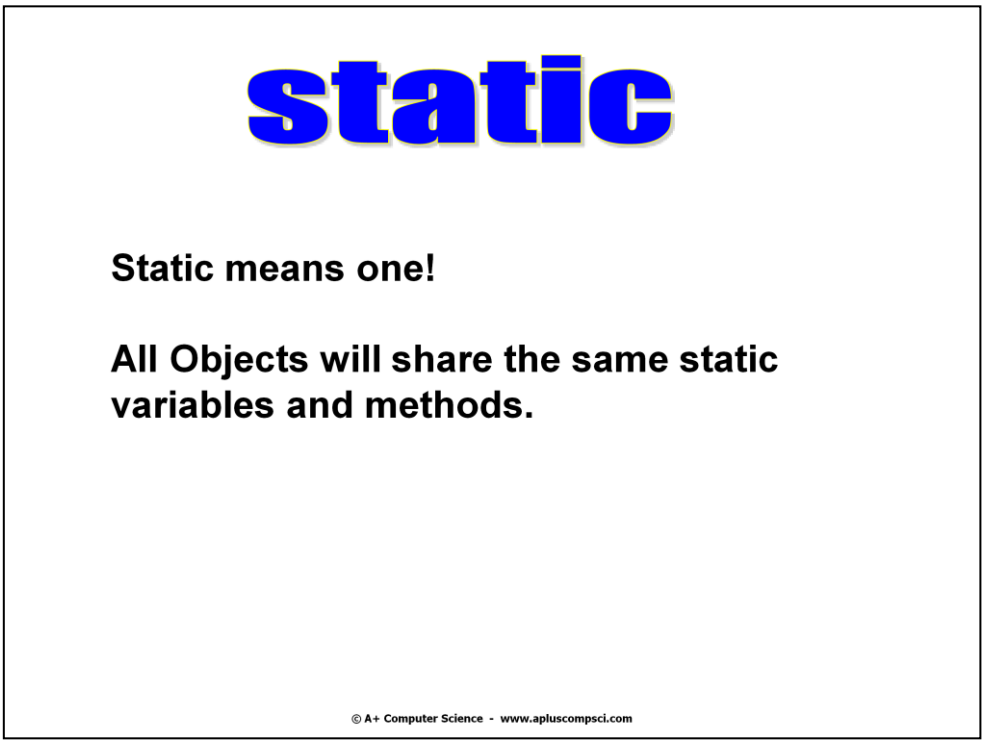
Static is a reserved word use to designate something that exists in a class.

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

Static variables and methods are bound to a class not an object instantiation. Static variables and methods can be used even if no objects of that class type have been instantiated.

`Math.ceil(7.5)` and `Arrays.sort(theRay)` are examples of static methods.

`System.out` is an example of a static reference variable.



For methods and variables labeled static, there will only be one.

The main method is a static method. There will only be one main method for a class.

static

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

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

all Monster share count

In the Monster class, count is a static variable. All instantiations of Monster share the same count variable. Count is being used to keep track of the number of Monsters that have been instantiated.

Each time the Monster constructor is called, the count variable is increased by one.

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 20