

Inheritance



© A+ Computer Science - www.apluscompsci.com

What is inheritance?

© A+ Computer Science - www.apluscompsci.com

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

© 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.

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.

© A+ Computer Science - www.apluscompsci.com

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.

Inheritance

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

A class can extend one other class.

Java does not support multiple inheritance.

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

© A+ Computer Science - www.apluscompsci.com

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 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);
```

inheritance example

OUTPUT

```
8  
8
```

© A+ Computer Science - www.apluscompsci.com

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 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 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());
```

inheritance example

OUTPUT

```
3  
2
```

© A+ Computer Science - www.apluscompsci.com

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.

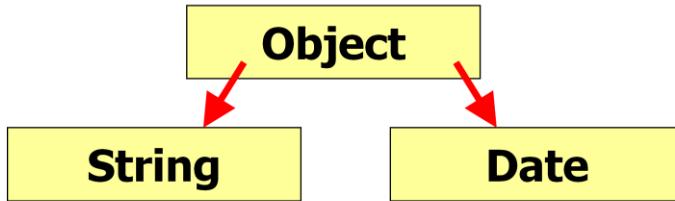
The second println prints out 2 because the setX method set x to 2.

**Open
inheritone.java**

© A+ Computer Science - www.apluscompsci.com

class Object

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



© A+ Computer Science - www.apluscompsci.com

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**

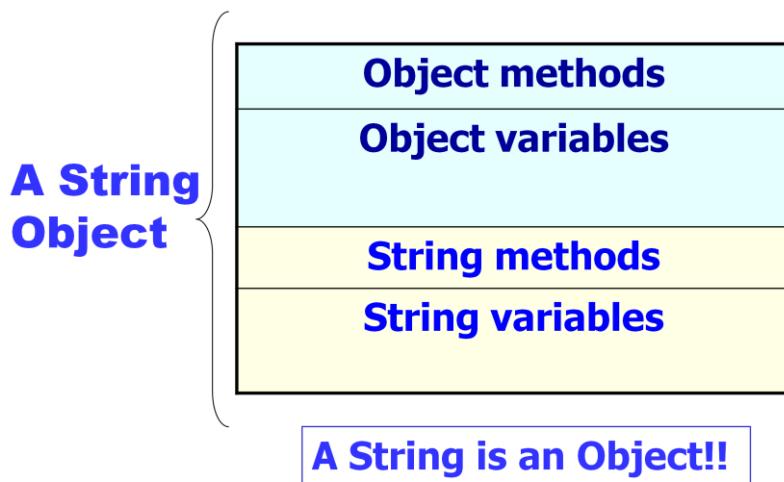


© A+ Computer Science - www.apluscompsci.com

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.

What's on the inside?



© A+ Computer Science - www.apluscompsci.com

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

© A+ Computer Science - www.apluscompsci.com

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**

© A+ Computer Science - www.apluscompsci.com

**Public
Protected
Private**

© A+ Computer Science - www.apluscompsci.com

public

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

© 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.

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.

© A+ Computer Science - www.apluscompsci.com

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.

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.

© A+ Computer Science - www.apluscompsci.com

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.

© A+ Computer Science - www.apluscompsci.com

Open
pubprotpriv.java

© A+ Computer Science - www.apluscompsci.com

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

© A+ Computer Science - www.apluscompsci.com

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.

```
class Monster
{
    private String myName;

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

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

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

this

calls Monster(name)

© A+ Computer Science - www.apluscompsci.com

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

super – refers to the parent class

super.toString(); legal

super.super.toString(); illegal

super(); parent default constructor call

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

© 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.

```

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;
    }
}

```

super this

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

super – refers to the parent

© A+ Computer Science - www.apluscompsci.com

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

© A+ Computer Science - www.apluscompsci.com

Start work on Lab 20

© A+ Computer Science - www.apluscompsci.com

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( ) }
}

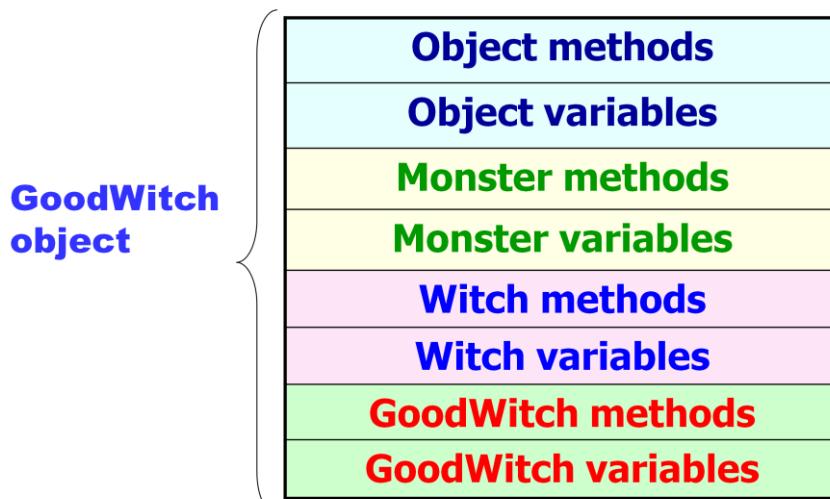
class GoodWitch extends Witch
{
    public GoodWitch() { //automatically calls super( ) }
}
```

© A+ Computer Science - www.apluscompsci.com

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.

What's on the inside?



© A+ Computer Science - www.apluscompsci.com

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.**

© A+ Computer Science - www.apluscompsci.com

Polymorphism

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

© A+ Computer Science - www.apluscompsci.com

Polymorphism

Object x = "compsci";

System.out.println(x);

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

OUTPUT
compsci

© A+ Computer Science - www.apluscompsci.com

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.

Polymorphism

Object x = "compsci";

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

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

OUTPUT
compsci

© A+ Computer Science - www.apluscompsci.com

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.

Polymorphism

```
Object x = "compsci";
```

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

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

OUTPUT

syntax error

© A+ Computer Science - www.apluscompsci.com

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 length () method; thus, Java reports a compile error.

Polymorphism

```
Object x = "compsci";
```

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

The cast will now let this code compile.

OUTPUT

7

© A+ Computer Science - www.apluscompsci.com

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

Polymorphism

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

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

Is this okay or not okay?

© A+ Computer Science - www.apluscompsci.com

A child can never refer to a parent.

Polymorphism

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

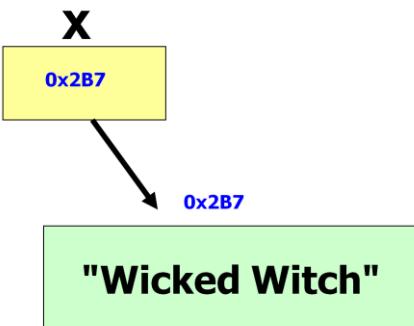
```
System.out.println(x);  
System.out.println(y);
```

Is this okay or not okay?

© A+ Computer Science - www.apluscompsci.com

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.

References



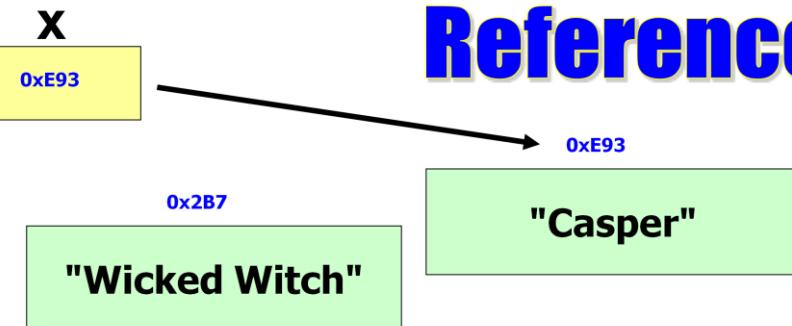
Monster x = new Witch("Wicked Witch");

Monster reference x refers to a Witch!

© A+ Computer Science - www.apluscompsci.com

X is referring to a Witch.

References



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

Monster reference x now refers to a Ghost!

© A+ Computer Science - www.apluscompsci.com

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

Open poly.java

© A+ Computer Science - www.apluscompsci.com

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.

© A+ Computer Science - www.apluscompsci.com

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 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");
    }
}

```

method override

© A+ Computer Science - www.apluscompsci.com

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

© A+ Computer Science - www.apluscompsci.com

method override

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

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

© A+ Computer Science - www.apluscompsci.com

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.

```

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( ) illegal – will not compile
    {
        System.out.println("overRide in Witch");
    }
}

```

© A+ Computer Science - www.apluscompsci.com

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

© A+ Computer Science - www.apluscompsci.com

What is composition?

© A+ Computer Science - www.apluscompsci.com

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

© 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?

© A+ Computer Science - www.apluscompsci.com

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**

© A+ Computer Science - www.apluscompsci.com

What is static?

© A+ Computer Science - www.apluscompsci.com

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.



© A+ Computer Science - www.apluscompsci.com

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.

static

Static means one!

All Objects will share the same static variables and methods.

© A+ Computer Science - www.apluscompsci.com

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; all Monster share count

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

© A+ Computer Science - www.apluscompsci.com

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

© A+ Computer Science - www.apluscompsci.com

KeyListener

Interface

© A+ Computer Science - www.apluscompsci.com

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;
```

© A+ Computer Science - www.apluscompsci.com

Continue work on Lab 20

© A+ Computer Science - www.apluscompsci.com