 Inheritance and Polymorphism

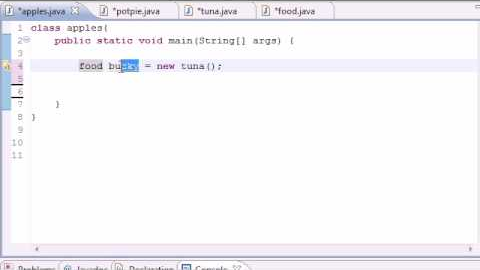
**Object-oriented programming allows you to define new classes from existing classes.**

**This is called inheritance.**

**Inheritance enables you to define a general class (i.e., a superclass) and later extend it**

**to more specialized classes (i.e., subclasses).**

[Java Programming Tutorial - 55 - Intoduction to Polymorphism](https://www.youtube.com/watch?v=0xw06loTm1k)



A **superclass** is also referred to as a **parent clas**s or a **base class**, and a **subclass** as a **child class**, an **extended clas**s, or a **derived class**.

Exteneded classes inherit data fileds and methods from the super class like children inherit traits from, their parents

public class Circle **extends** GeometricObject // To make the class Ccircle an // //extended or child of the superclass or parent class called GeometricObject. //Much easier in Java

*This variable seems to be the hidden argument for all methods in a class*

*This.variable not needed when variables not inintialized in definition*

Java only has single inheritance. Child only have 1 parent. There may be ways to circumvent this in Java

**The keyword super refers to the superclass and can be used to invoke the superclass’s methods and constructors.**

**A subclass inherits accessible data fields**

The **super** keyword can call a super class constructor or a super class *method*

super(), or super(parameters); // Syntax to call super class constructor

super.method(parameters); // Syntax to call a super class method

Subclass constructor 1st invokes its superclass constructor before doing anything else. If a super method not explicitly defined in constructor, it is automatically defined by default; it calls the default super class constructor. If there are really extended classes. Classes that are derived from derived classes constructor chaining can happen this call of super will lead to constructor chaining.

You must use the keyword super to call the superclass constructor, and the call must be the first statement in the constructor.

*What if super class has a constructor with multiple parameters but super(parameters) no called in child?*

*Super classses should only have no arg constructors?*

**To override a method, the method must be defined in the subclass using the same signature**

**and the same return type as in its superclass.**

An instance method can be overridden only if it is accessible. Thus a private method cannot

be overridden, because it is not accessible outside its own class. If a method defined

in a subclass is private in its superclass, the two methods are completely unrelated.

■ Like an instance method, a static method can be inherited. However, a static method

cannot be overridden. If a static method defined in the superclass is redefined in a

subclass, the method defined in the superclass is hidden. The hidden static methods

can be invoked using the syntax SuperClassName.staticMethodName.

Super class method that are overidden can't be accessed by the subclasses that overode them

**Overloading means to define multiple methods with the same name but different signatures.**

**Overriding means to provide a new implementation for a method in the subclass.**

To avoid mistakes, you can use a special Java syntax, called **override annotation**, to place

@Override

public class CircleFromSimpleGeometricObject

extends SimpleGeometricObject {

// Other methods are omitted

@Override // Syntax error will result if toString() doesn't overide a method of the super class error result b/c @Override

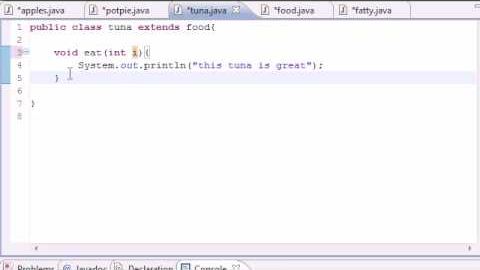
public String toString() {

return super.toString() + "\nradius is " + radius;

}

}

[Java Programming Tutorial - 57 - Overriding Rules](https://www.youtube.com/watch?v=zN9pKULyoj4)



**Every class in Java is descended from the java.lang.Object class.**

If no inheritance is specified when a class is defined, the superclass of the class is **Object** by

default.

**Polymorphism means that a variable of a supertype can refer to a subtype object.**

A

type defined by a **subclass** ( child class) is called a **subtype**, and a type defined by its superclass is called

a **supertype**. Classes are essentially definitions of data types. The class name used in a program is a data type

**A method can be implemented in several classes along the inheritance chain. The JVM decides which method is invoked at runtime.**

The type that declares a variable is called the variable’s **declared type**.

The declared type of the reference variable decides which method to match at compile time.

The **actual type** of the variable is the actual class for the object referenced by the variable.

The JVM dynamically binds the implementation of the method at runtime, decided by the actual type of the variable.

Object o = new GeometricObject(); // declared type of o is Object

System.out.println(o.toString()); // actual type of the variable is the object's //referenced variable

**One object reference can be typecast into another object reference. This is called casting object.**

m(new Student()); // equivalent to

Object o = new Student(); // Implicit casting // a Student is an object. Right side //object. Left side reference variable

m(o);

Instancce of student is an instant of the object. Object is the superclass of all classes

*Can Student be assgined an object class. Does implicit casting occcur only when a superclass reference variable references an extended class of it?*

*No. Animal is a superclass ood Dog. An Animal is not nessarily a Dog but a Dog is an animal. Implicit casting occcures when a superclass reference variable references an extended class of it?*

Object o = new Student(); // Imlicit casting. A student is an object. It is upcasted //to be a superclass objecct

Student b = new Object(); // No implicit casting. An object is not nesecarily a //student. Syntax error

Student b = (Student)o; // Explicit casting. Object ref var o downcast to be the //subclass object Student

The right side refers to the type of referencce variable. The left side refers to the actual object. Hence you can have anonymous object, which only last for one command.

It is always possible to cast an instance of a subclass to a variable of a superclass (known as

**upcasting**), because an instance of a subclass is always an instance of its superclass. *I*

*The instance of the superclass is always in the instance of a subclass?*

When casting an instance of a superclass to a variable of its subclass (known as **downcasting**),

Explicit casting is needed for this. **Syntax error will result if expliccit casting used to upcast**.

Object to be casted should be an instance of the subclass. EX:

Student b = (Student)o; // object is an instance of the subclass Student class

Object o = new Student(); // Student is an instance of the superclass object not //the subclass student. Naturally subclasses have instances of both superclass and //and subclass but not when casted to a superccclass.

Superclass object to subclass object good for explicit casting otherwise it's not. If superclass is instance of subclass explicit casting otherwise not.

When reading asignment operators read data ccomming "in" to it from right to left.

The right side is the object to b casted.

Object.tostring

<https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>

**ClassCastException** occurs explicit casting doesn't work

Use **instanceof** :Keyword returns a 1 if an object is an instance of another object. It's like the @Override keyword, causes a compiler error if method after it does not overide.

Object myObject = new Circle();

... // Some lines of code

/\*\* Perform casting if myObject is an instance of Circle \*/

if (myObject instanceof Circle) {

System.out.println("The circle diameter is " +

((Circle)myObject).getDiameter());

...

}

*What is so special of static methods. Yes they can only work with static variables and other static methods*

Explicit casting is used to enable generic programming making variables accept an object of a subtype

Explicit ccasting is used to demote variables. In order to demote variables you need to cast then expliccitely unlike when you are demoted variables.

<http://www.java2s.com/Tutorial/SCJP/0080__Type-Casting/Implicitandexplicitcasting.htm>

Object o = new Student(); // Imlicit casting. A student is an object. It is upcasted //to be a superclass objecct

Student b = new Object(); // No implicit casting. An object is not nesecarily a //student. Syntax error

Student b = (Student)o; // Explicit casting. Object ref var o downcast to be the //subclass object Student

Superclasses can be expliccitely ccasted to subcclasses but subclasses can only be implicitely casted to superclasses even though the subclass is bigger and has the instancce of the superclass. A superclass and an subclass is like a double to an int where the int has mucch more memory and ccan store numbers than the double.

When you assign objects with eacch other, are your ethods and data fields replaced with the object you assign the other object to. Is it added to the datafields and methods the left object had.

Object o = new Student(); // Wouldn't this be usely if object was staticcally bounded

o.toString()

/\* But it is naturally not. The JVM will go search for toString() starting with the class Student. If such a method is not located in Student its supercclass and superclasses of superclasses are searcched for the method ending with the object class.

*What if you invoke a method that is not found in any of the sub and super classes?* \*/

int age = 45;

byte newAge = (byte)age; // A new value is assigned to newAge

Object o = new Circle();

Circle c = (Circle)o; // No new object is created

**Like the toString() method, the equals(Object) method is another useful method defined in the Object class.**

The **equals()** method provided by Object tests whether the object *references* are equal—that is, if the objects compared are the exact same object.

From <[*https://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html*](https://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html)>

object1.equals(object2); // Syntax for equal method

This implementation checks whether two reference variables point to the same object using the == operator.