Object Superclass

The **Object** class is the superclass of all other classes in Java and a part of the built-in `java.lang` package. If a parent class isn't specified using the **extends** keyword, the class will inherit from the `Object` class. What does a class inherit from the `Object` class? There are two main methods that are most used, `toString()` and `equals(Object)`, from the `Object` class:

- `String toString()`
- `boolean equals(Object otherObjectToCompareTo)`

For a quick visual primer on how the `Object` class is the superclass of all other Java classes, take a look at this video here. Notice how he uses his IDE to see what methods are available–this is a common developer trick that keeps us from having to memorize code. See if you can't take advantage of it yourself:
Embed anything (PDFs, Google Maps, Spotify, etc...)

https://youtu.be/sHXeW0t9S1g

#The `toString()` method

One commonly overridden Object method is `toString()`, which is often used to print out the attributes of an object. It is a good idea to write your own `toString()` method in every class you create. When you don't specify what the `toString()` method should do, it just gives you the name of the object and its place in memory when you run the `toString()` command on it. This, generally speaking, is not very useful to us. Therefore, it's a good idea to define your own `toString()` method that gives you the details you actually need. In a subclass, `toString()` can call the superclass's `toString()` method using `super.toString()` and then add on its own attributes.

#Coding Exercise

In the following code, the `Person` class overrides the `Object`'s `toString()` method and the `Student` class overrides the `Person`'s `toString()` method. Each class adds on its own special attributes.

After trying the code below, add another subclass called `APStudent` that extends `Student` with a new attribute called `apScore`. Then create the parameterized constructor in the `APStudent` class(use super keyword to call the parent constructor), override its `toString()` method to call the superclass method and then add on the `apScore` in the returned string. Create an `APStudent` object in the `main` method to test it. **Test it using a value of 90 as the `APStudent`'s `apScore` while keeping other attributes similar to those of the `student` object.**

You might also want to see what the `toString()` method does when we don't offer the class its own implementation of the method. Try deleting the specified toString methods and see what comes out. Note: if this is too advanced for you at the moment, check the Extra Resources below for a similarly worked problem in YouTube format.

3

Run

Student.java Person.java APStudent.java

```
class Student extends Person {
    protected int id;
    public Student(String name, int id) {
        super(name);
        this.id = id;
    }
}
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String toString() {
        return name;
    }

    public static void main(String[] args) {
        APStudent aps = new APStudent("Tully", 1001, 90);
        System.out.println(aps); // call APStudent toString method and print o
utput on the console
    }
}
```

```
public class APStudent extends Student {
    private int apScore;

    public APStudent(String name, int id, int apScore) {
        super(name, id);
        this.apScore = apScore;
    }

    @Override
    public String toString() {
        return super.toString() + " " + id + " " + apScore;
    }
}
```

#The `equals()` Method

One of the other important things that gets inherited from the Object superclass is the `equals(Object obj)` method. This method is used to test if the current object and the passed object (called `obj` here) are "equal". But what does that mean?

As seen in the code below, the `equals` method that is inherited from the `Object` class only returns true if the two objects references refer to the exact same object.

#**Coding Exercise**

Try to guess what the code below will print out before running it. Write what you think will happen in the space below:
false false false true

Run

Person.java

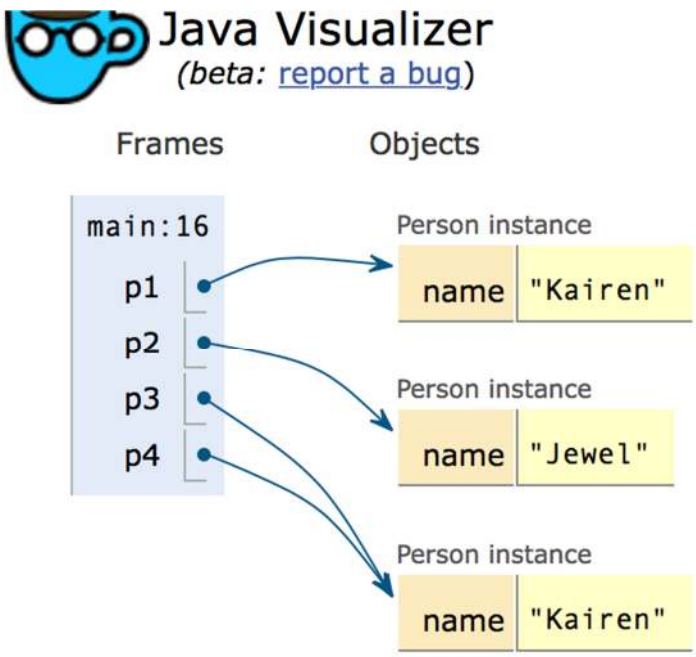
```
public class Person {
   private String name;

   public Person(String theName) {
      this.name = theName;
   }
```

```
    public static void main(String[] args) {
        Person p1 = new Person("Kairen");
        Person p2 = new Person("Jewel");
        Person p3 = new Person("Kairen");
        Person p4 = p3;
        System.out.println(p1.equals(p2));
        System.out.println(p2.equals(p3));
        System.out.println(p1.equals(p3));
        System.out.println(p3.equals(p4));

    }
}
```

The `equals` method inherited from the `Object` class only returns `true` when the two references point to the same object as shown in the code above and the image below.



A picture from the Java Visualizer showing that only p3 and p4 refer to the same object.

#Overriding the equals Method

If you want to change how the inherited `equals` method works you can **override** it so that the new method is called instead of the inherited one. The `String` class **overrides** the inherited equals method to return true when the two objects have the same characters in the same order as shown in the code below.

#Coding Exercise

Try to guess what the code below will print out before running it. Write what you think will happen in the space below:

false false true

Run

StringTest.java

```
public class StringTest {
   public static void main(String[] args) {
      String s1 = "hi";
      String s2 = "Hi";
      String s3 = new String("hi");
      System.out.println(s1.equals(s2));
      System.out.println(s2.equals(s3));
      System.out.println(s1.equals(s3));
   }
}
```

CONSOLESHELL

Any class can override the inherited `equals` method by providing a method with the same method signature (method name and parameter list) and return type. The provided method will be called instead of the inherited one, which is why we say that the new method **overrides** the inherited method. The `Person` class below **overrides** the inherited `equals` method.

#Coding Exercise

Try to guess what the code below will print out before running it. Write what you think will happen in the space below:

false false true true

Run

Person.java

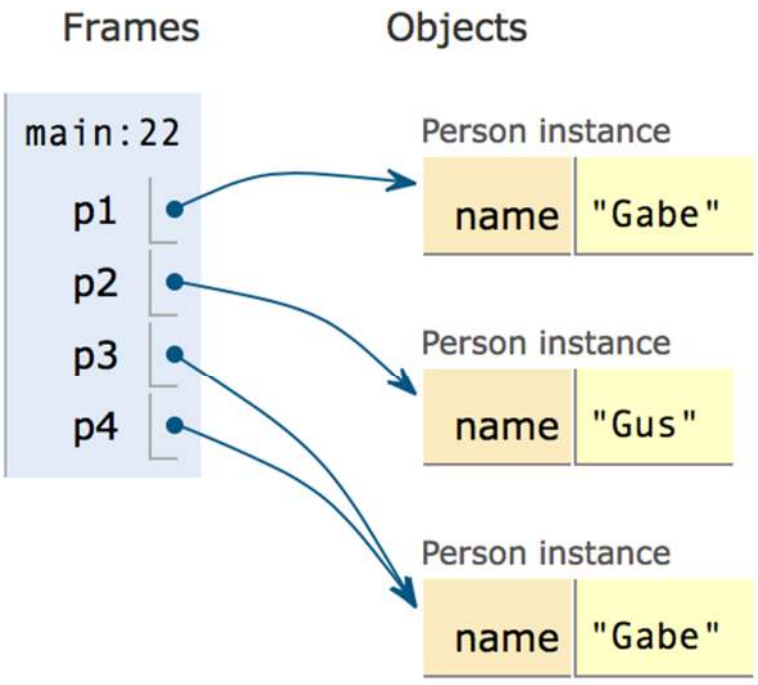
```
public class Person {
   private String name;

   public Person(String theName) {
```

```
        this.name = theName;
    }

    /** Overriden equals method that checks if names are equal
        in this Person object and an the other Object.
        */
    public boolean equals(Object other) {
        // Type cast other to a Person
        Person otherPerson = (Person) other;
        // Check if names are equal
        return this.name.equals(otherPerson.name);
    }

    public static void main(String[] args) {
        Person p1 = new Person("Gabe");
        Person p2 = new Person("Gus");
        Person p3 = new Person("Gabe");
        Person p4 = p3;
        System.out.println(p1.equals(p2));
        System.out.println(p2.equals(p3));
        System.out.println(p1.equals(p3));
        System.out.println(p3.equals(p4));
    }
}
```



A picture from the Java Visualizer showing the object references and objects.

If you're curious, you can step through this code in the Java Visualizer by clicking on the following link: OverrideEquals Ex.

To write your own equals method, you must:

1. Use the `public boolean equals(Object other)` method signature
2. Type cast other to your Classname
3. Return whether this object's attribute(s) equals the other object's attribute(s) with `==` for primitive types like int and double, or `equals` for reference types like Strings or another class.

InfoWarningTip

Type casting sounds complicated, but it's fairly simple. See an example tutorial here.

```
public boolean equals(Object other) {
// Type cast other to your Classname
Classname otherObj = (Classname) other;
// Check if attributes are equal
return (this.attribute == otherObj.attribute);
// or this.attribute.equals(otherObj.attribute) if attribute a
String
}
```

If you need to check multiple attributes, for example a name and an address for Person objects, you can use && to combine the tests:

```
return (this.attribute1 == otherObj.attribute1) &&
this.attribute2.equals(otherObj.attribute2)
```

If you are writing an equals method for a subclass, you can call the superclass equals using the **super** keyword to check the attributes in the superclass and then check the attributes in the subclass.

```
return super.equals(otherObj) &&
(this.attribute == otherObj.attribute)
```

#Programming Challenge : Savings Account

In the following code, a bank `Account` class contains the account holder's `name` and the money `balance` in the account.

Work in pairs within your teams to write the following code and test each part before moving on to the next step:

1. Write a `toString()` method for `Account` that returns the `name` and `balance` with a comma in between.
2. Write an `equals` method for `Account` that takes in two Account objects and checks that their `name` and `balance` attributes are equal.
3. In a subclass called `SavingsAccount` (see the separate file below) extend `Account` and declare an `interest` rate variable.
4. Write a `toString()` method for `SavingsAccount` that returns a call to the super `toString()` method and the `interest` rate with a comma in between.
5. Write an equals method for `SavingsAccount` that calls the superclass `equals` method and checks that the `interest` rates are equal.

Complete the subclass SavingsAccount below which inherits from `Account` and adds an `interest` rate variable. Write a `toString` and an `equals` method for it.

2

Run

Account.java SavingsAccount.java

```
public class Account {
    private String name;
    private double balance;

    public Account(String name, double balance) {
       this.name = name;
       this.balance = balance;
    }

    //Step 1 and Step 2
     public String toString() {
         return name + ", " + balance;
    }

    public boolean equals(Account otherAccount) {
         return name.equals(otherAccount.name) && balance == otherAccount.balan
ce;
    }

    public static void main(String[] args) {
```

```
        // Code to test classes
        Account a1 = new Account("Illa", 500000);
        Account a2 = new Account("Javier", 10000);
        Account a3 = new Account("Javier", 50000);
        Account a4 = new Account("Illa", 50000);

        // toString Account
        System.out.println("toString() Account");
        System.out.println("a1 " + a1);
        System.out.println("a2 " + a2);

        System.out.println("\nequals() Account");
        // equals Account
        System.out.println("a2 and a3 are equal? " + a2.equals(a3));
        System.out.println("a1 and a4 are equal? " + a1.equals(a4));

        SavingsAccount sa1 = new SavingsAccount("Illa", 500000, 0.5);
        SavingsAccount sa2 = new SavingsAccount("Javier", 10000, 0.3);
        SavingsAccount sa3 = new SavingsAccount("John", 500000, 0.5);
        SavingsAccount sa4 = new SavingsAccount("Illa", 500000, 0.5);

        System.out.println("\ntoString() Savings account");
        // toString Savings account
        System.out.println("sa1 " + sa1);
        System.out.println("sa2 " + sa2);
        System.out.println("sa3 " + sa3);
        System.out.println("sa4 " + sa4);

        System.out.println("\nequals() Savings account");
        // equals Savings account
        System.out.println("sa1 and sa2 are equal? " + sa1.equals(sa2));
        System.out.println("sa1 and sa3 are equal? " + sa1.equals(sa3));
        System.out.println("sa2 and sa3 are equal? " + sa2.equals(sa3));
        System.out.println("sa1 and sa4 are equal? " + sa1.equals(sa4));
    }
}
class SavingsAccount extends Account {
    private double interestRate;

    public SavingsAccount(String name, double balance, double interestRate) {
        super(name, balance);
        this.interestRate = interestRate;
    }

    @Override
    public String toString() {
        return super.toString() + ", " + interestRate;
    }
```

```
    @Override
    public boolean equals(Account otherAccount) {
        if (otherAccount instanceof SavingsAccount) {
            SavingsAccount otherSavingsAccount = (SavingsAccount) otherAccount
;
            return super.equals(otherAccount) && interestRate == otherSavingsA
ccount.interestRate;
        }
        return false;
    }
}
```

#Summary

- The Object class is the superclass of all other classes in Java and a part of the built-in `java.lang` package.
- The following Object class methods and constructors, including what they do and when they are used, are part of the Java Quick Reference:
  - `String toString()`
  - `boolean equals(Object other)`
- Subclasses of Object often override the `equals` and `toString` methods with class-specific implementations.

#Extra Resources

This video walks you through some basics for defining a class and then looks at how the `toString()` method works in its default state. If you're a beginner, this is a great walkthrough, but feel free to skip a minute or two into the video if you don't need help setting up your files:
Embed anything (PDFs, Google Maps, Spotify, etc...)
https://youtu.be/d08oJlwVgyo

We mentioned typecasting in this lesson, too. Why not hear from the same friendly face about it? He talks about some of the concerns that come along with the process, so it's a good idea to hear him out:

Embed anything (PDFs, Google Maps, Spotify, etc...)
https://youtu.be/H0LNjF9PSeM