



Fundamental Methods

Important common methods

Methods Inherited from Object

- ❑ Every class inherits methods from Object.
- ❑ Some methods are key to object behavior

`java.lang.Object`

```
#clone()      : Object
+equals(Object) : bool
+finalize()   : void
+getClass()   : Class
+hashCode()   : int
+toString()   : String
+wait()       : void
```

How the Methods are Used

toString() - implicitly invoked whenever Java needs to display (or copy) object as a String:

```
System.out.println( x ); // calls x.toString()
```

```
String greet = "Hello, "+person; // person.toString()
```

equals(Object other) - test for equality. Used by
List.contains(something), List.indexOf(something).

```
List<Course> courses = Registrar.getMyCourses();
```

```
Course prog1 = new Course("01219114","Prog 1",3);
```

```
if ( courses.contains(prog1) ) ...
```

toString()

Most classes should define their own `toString()` method.

Exceptions:

- inherits a usable `toString()` from a parent class
- object is not intended to be printed; e.g. controllers, UI classes, "transport objects", utility classes like `Math` or `Arrays`.

`@Override` (annotation) is optional. Used by compiler to detect accidental misspelling.

```
public class MenuItem {  
    @Override  
    public String toString() {  
        return itemName;  
    }  
}
```

Course without "equals"

```
public class Course {  
    private final String id;  
    private String name;  
    private int credits;  
    public Course(String id, String name, int cred) {  
        this.id = id;  
        this.name = name;  
        this.credits = cred;  
    }  
    ... get/set methods, but no "equals"
```

final means you cannot change the value after it is set the first time.
final attributes must be set in a constructor.

Use of "equals"

Course does **not** define "equals" method,
so it **inherits** "equals" from Object.

What does object.equals() method do?

```
Course c1 = new Course("01219114", "Programming", 3);  
Course c2 = c1;  
System.out.println(c1 == c2); // true  
System.out.println(c1.equals(c2)); // also true  
// but...  
c2 = new Course("01219114", "Programming", 3); //same!  
System.out.println(c1.equals(c2)); // false
```

Object.equals() is just ==

The Object **equals** method is same as ==

This is (usually) **not** what we want.

```
public class Object {  
  
    public boolean equals(Object obj) {  
        return this == obj;  
    }  
}
```

Collections (List, Set) use equals

```
List<Course> courselist = new ArrayList<Course>();  
Course c1 = new Course("01219114", "Programming", 3);  
Course c2 = new Course("01219114", "Programming", 3);  
  
courselist.add( c1 );  
  
// what courses have I enrolled in?  
courselist.contains( c1 )    // true  
courselist.contains( c2 )    // false
```


When *should* 2 Courses be equal?

1. Depends on the application.
2. Should be clearly defined and **documented**.

Course Enrollment Application

- a department might *change* the name of a course.
- Registrar relies on course ID to decide if student has taken a course, assigning grades, prerequisites, etc.

When *should* 2 Courses be equal?

Course Enrollment Application:

- a department might change the name of a course.
- Registrar relies on course ID to decide if student has taken a course, assigning grades, prerequisites, etc.

Therefore (design decision):

Two courses are **equal** if **the id is same** (even if String name is different).

Writing equals()

```
public class Course {  
    /** Two courses are equal if they have same id.  
     */  
    @Override  
    public boolean equals(Object obj) {  
        if (obj == null) return false;  
        if (obj.getClass() != this.getClass())  
            return false;  
        // cast it to Course so we can get attributes  
        Course other = (Course)obj;  
        // Finally! compare course IDs (as Strings)  
        return this.id.equals( other.getId() );  
    }  
}
```

4-Step Template for equals()

@Override

Must be "Object" not Course

```
public boolean equals(Object obj) {  
    if (obj == null) return false;           //1  
    if (obj.getClass() != this.getClass())    //2  
        return false;  
    // cast to this class so we can get attributes  
    Course other = (Course)obj;               //3  
    // Finally! compare this and other the way your  
    // application wants.  
    return _____;                       //4  
}
```

4-Step Template explained

@Override

Must be "Object" not Course

```
public boolean equals(Object obj) {
```

1. Check that argument is not null

```
if (obj == null) return false;
```

2. Argument must be same class as this class

```
if (obj.getClass() != this.getClass())
```

```
    return false;
```

3. Cast to this class so we can get attributes

```
Course other = (Course)obj;
```

4. Compare this and other as your app requires

```
return this.id.equals( other.getId() );
```

```
}
```

Why are these 4 steps necessary?

@Override

```
public boolean equals(Object obj) {
```

1. Required to avoid NullPointerException later

```
if (obj == null) return false;
```

2. Can't compare Course & Dog or Course & String

```
if (obj.getClass() != this.getClass())
```

```
    return false;
```

3. "Object" doesn't have attributes of a Course

```
Course other = (Course)obj;
```

4. Domain logic. Why we wrote this method!

```
return this.id.equals( other.getId() );
```

```
}
```

Find 4 Errors

```
public class Course {  
    private final String id;  
  
    @Override  
    public boolean equal(Object obj) {  
        if (obj.getClass() != this.getClass())  
            return false;  
        if (obj == null) return false;  
        Course other = (Course)obj;  
        // compare course IDs (Strings)  
        return this.id == obj.id;  
    }  
}
```

Find the Errors, again

```
public class Course {  
    private final String id;  
  
    public boolean equals(Course obj) {  
        if (obj == null) return false;  
        Course other = (Course)obj;  
        // compare course IDs (Strings)  
        return this.equals(other.getId());  
    }  
}
```


Don't write nested if - Points deducted

1. Harder to follow the logic.
2. Possible "dangling else" error.

```
@Override
```

```
public boolean equals(Object obj) {  
    boolean check = false; // no var named "check"!  
    if (obj != null) {  
        if (obj.getClass() == this.getClass()) {  
            Course other = (Course)obj;  
            if ( this.id.equals( other.getId() )  
                check = true;  
        }  
    }  
    return check;  
}
```

Practice - write equals

On **paper** or in **an editor**, write **equals** for the Money class.

Two Money objects are equal if the amount **and** currency are same.

```
public class Money {  
    private String currency  
    private double amount;  
    /**  
     * Money objects are equal if & only if  
     * the currency and amount are the same.  
     */  
    public boolean equals(Object obj) {  
        //TODO  
    }  
}
```

Solution

```
/**
 * Money objects are equal if & only if
 * the currency and amount are the same.
 */
public boolean equals(Object obj) {
    //TODO
}
```

A variation on equals

Sometimes it makes sense for objects of different classes to be "equal". (Don't write this on exam, unless specified.)

```
@Override
```

```
public boolean equals(Object obj) {
```

```
    1&2. obj is not null and an instance of  
        this class or a subclass of this class
```

```
    if ( !(obj instanceof Course) ) return false;
```

```
    3. Cast it to our class to access the attributes
```

```
    Course other = (Course)obj;
```

```
    4. Domain logic: When are two objects equal?
```

```
    return this.id.equals( other.getId() );
```

```
}
```

Other Important Methods to Know

int hashCode () - hash of object data, used by HashSet, HashMap, and some other collections. Should be *consistent* with equals:

`a.equals(b) => a.hashCode() == b.hashCode()`

but not the converse.

clone () - make a *deep copy* of an object.
This is covered in OOP2.

Reference

Big Java, 5E

Oracle Java Tutorial

- `toString()`, `equals()`, `Object` class