**Recitation 0: Equals Method, Clone Method, Documentation**

*Making Equals Easy*

* Using the “==” operator
  + Compares *Addresses* NOT *objects*
  + Example 1:

Person person1 = **new** Person(**"bob"**,42,**"123-45-1234"**);

Person person2 = **new** Person(**"Mary"**,58,**"111-23-4456"**);

Person person3 = **new** Person(**"bob"**,42,**"123-45-1234"**);

Address1

Address2

Address3

Name = “Mary”

Age = 58

ssn = “111-23-4456”

Person2

Name = “bob”

Age = 42

ssn = “123-45-1234”

Person3

Person1

Name = “bob”

Age = 42

ssn = “123-45-1234”

A Distinct Person

Object

A Distinct Person

Object

A Distinct Person

Object

What is the result of: person1 == person3

Answer: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Why?

How do we check for object Equality?

* The pseudo-code of the equals method:

public boolean equals(Object obj){

If(*obj is actually a person*){ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Person p = (Person)obj; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*return true if all fields are equal*

}

return false;

}

* \*\*\*Typecasting\*\*\*\*
  + The equals method accepts an \_\_\_\_\_\_\_\_\_\_ as an argument
  + To tell the compiler that the object is actually a “Person” we typecast
  + EX: Person p = (Person) obj;

The variable name for the cast object

we want

The Type object we need

The variable name for the object we have

The Type of object we need

The full person equals method:

*Clarifying the Clone Method*

* The clone method:

public Person clone(){

Person newPerson = new Person(this.name,this.age,this.ssn);

return newPerson;

}

* + Shallow copy:
    - Lets say we add a new field to Person
      * private Date birthdate
        + Which is an object that represents a person’s birthday
      * What happens when we try:

public Person clone(){

Person newPerson = new Person(this.name,this.age,this.ssn, this.birthdate);

return newPerson;

}

Address2

Person1 copy

Name = “bob”

Age = 42

ssn = “123-45-1234”

Phone

Address1

Person1

Name = “bob”

Age = 42

ssn = “123-45-1234”

Phone

A Distinct Person

Object

SAME Date Object

“11/15/97”

A Distinct Person

Object

* This will create a SHALLOW copy: if we edit the birthday of one person, the other persons birthday will also be edited
* How Do we fix this? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* String do not have to be deep copied because they are immutable (cannot be altered once created)!

public Person clone(){

Person newPerson = new Person(this.name,this.age,this.ssn, this.birthdate.clone);

return newPerson;

}

*Creating deep Clone (and Equals) methods*

* To create a “deep” clone (or equals) method you use the cloning (or equals) method of one object inside of the cloning (or equals) method of another object.
  + Methods with many lines are hard to read and debug
  + How can we shorten clone and equals method?
  + Cloning example:
    - Let’s say we have a UniversityClass object which has a private field Person[] roster
    - We could do:

public Class clone(){

Class classCopy = new Class();

for(Person p: roster){

Person newPerson = new Person(p.getName(),p.getAge(),p.getSSN());

//THIS MAKES US USE TOO MANY GETTERS WHICH IS UNESSECCERY

classCopy.add(newPerson);

}

Return classCopy;

}

* + - Instead we will do:

public Class clone(){

Class classCopy = new Class();

for(person p in the class){

classCopy.add(p.clone()); MUCH SHORTER! EASIER TO READ/DEBUG!

}

Return classCopy;

}

*Discovering Documentation*

* API: Application Programmer Interface
  + How can we use the code someone else wrote?
* How can we document how to use our code so that other programmers can use it?
  + *Javadoc*
    - Javadoc creates a set style in which to write documentation so different programmers can communicate how their code works to each other

|  |  |  |
| --- | --- | --- |
| Tag | Meaning | Place |
| @see | See related content | Class, Method |
| @author | Author of the class | Class |
| @version | The version of the class (Used for updates to code) | Class |
| **@param** | Information on the parameter of a method | Method |
| **@return** | Information of the return value for a method | Method |
| **@exception** | Information on exceptions thrown by a method | Method |
| **@throws** | Information on exceptions thrown by a method | Method |
| @deprecated | Marks an element as deprecated | Class, Method |
| @since | The API version this element was first included | Class, Method |

\*Bold tags are the most important ones you will need to know for CSE 214

*Documentation Examples*

*/\*\*  
 \* This class Represents a person which has a name, age and associated SSN  
 \*  
 \** ***@author*** *Juan Tarquino  
 \*/***public class** Person {

………

}

*/\*\*  
 \* This method adds two positive numbers together  
 \*  
 \** ***@param num1*** *\* The first number to be added  
 \** ***@param num2*** *\* The second number to be added  
 \*  
 \** ***@return*** *\*The sum of the first and second number  
 \*  
 \** ***@throws*** *IllegalArgumentException*

*\* when either of the numbers is negative  
 \*/***public int** add (**int** num1, **int** num2) **throws** IllegalArgumentException{  
 **if**(num1 < 0 || num2 < 0)  
 **throw new** IllegalArgumentException(**"One of the numbers is negative!"**);  
  
 **return** num1 + num2;  
}

*/\*\*  
 \* This is a Constructor used to create a new Person object  
 \*  
 \** ***@param name*** *\* The name of the person  
 \** ***@param age*** *\* The age of the Person  
 \** ***@param ssn*** *\* The social security number of the person  
 \*/***public** Person(String name, **int** age, String ssn){  
 **this**.name = name;  
 **this**.age = age;  
 **this**.ssn = ssn;  
}