School of Computer Science

UNIVERSIDAD DE OVIEDO

Final Tips

Introduction to Programming

Academic year 2023-2024

Attributes / Local variables

```
import java.util.ArrayList;

class PersonalOrganizer {

  private ArrayList<String> notes;

  public PersonalOrganizer()  {
    ArrayList<String> notes = new ArrayList<String>();
  }

  public void addNote(String note) {
    notes.add(note);
  }
}
```

Comparing String objects

```
/**
 * Compares the name atribute of the class with the parameter
 * @param   name  name to be compared
 * return true if both names are equal and false in they are not
*/
public boolean isTheSameName(String name) {
  return this.name==name;
  return this.name.equals(name);
  return this.name.compareTo(name) == 0;
  return this.name.compareToIgnoreCase(name) == 0;
 }
```

You are comparing the reference, not the content!

Correct. Equals method is overridden in the String class

Correct. This method compares the content of the strings

Correct. This method compares the content of the strings (ignoring their casing)

Comparing objects

```
/**  VERSION ONE
 * Compares the day, month and year of the Date object with the
parameter (get methods are public)
 * @param   date  date to be compared
 * return true if both dates are equal and date in they are not
*/
public boolean equals(Date date) {
      return this.day ==date.getDate() &&
             this.month==date.getMonth() &&
             this.year==date.getYear() ;
 }
```

This method is not overriding the equals of the Object class. You can use it in your classes, but in the tests the assertEquals will not use it.

Comparing objects

```
/**  VERSION TWO
 * Compares the day, month and year of the Date object with the
parameter (get methods are public)
 * @param   date  date to be compared
 * return true if both dates are equal and date in they are not
*/

@override
public boolean equals(Object date) {
      Date date1=(Date) date;
      return this.day ==date1.getDate() &&
             this.month==date1.getMonth() &&
             this.year==date1.getYear() ;
 }
```

Parameter passing

- All parameters are passed by value (the value of the actual parameter is copied into the formal parameter). So, if we use a variable as the actual parameter its value cannot be change inside the method:

```
public void swap(int a, int b){
  int aux;
  aux = a;
  a = b;
  b = aux;
}
```

```
public void print() {
  int x = 2;
  int y = 4;
  swap(x,y);
  System.out.println(x + " " +  y);
}
```

Output: “2 4”

Parameter passing. References

- The previous explanation is true even if the parameters are references. But the objects pointed by the reference can be modified inside the method.

```
public void add(ArrayList <Person> list) {
  list.add(new Person());
}
public void m1() {
 ArrayList <Person> l=new  ArrayList<Person>();
 System.out.print (l.size()+” “);
 add(l);
 System.out.println(l.size());
}
```

Output: “0 1”

Invoking methods

- All the *normal* methods (but the static ones) must be invoked *on* an object. So you must be sure that the object exist before invoking a method on it:

```
public boolean isMale(Person p){
      return p.getGender()==Person.GENDER_MALE;
}
```

What if p is null? A Null Pointer Exception will be trown

Invoking methods

- All the *normal* methods (but the static ones) must be invoked *on* an object. So you must be sure that the object exist before invoking a method on it:

```
public boolean isMale(Person p){
      if (p==null) {
        // Do what must be done in this case
      } else {
            return p.getGender()==Person.GENDER_MALE;
        }
}
```

Invoking methods

- You must be specially careful when working with data structures (arrays, arrayLists, …) that can contain null elements:

```
public int numberOfChildren(Person [] list){
      int count;
      for (Person p:list) {
            if (p!=null && p.getAge()<10)) {
                  count++;
            }
      return count;
}
```