G51PRG: Introduction to Programming Second semester

Lecture 2

Natasha Alechina
School of Computer Science & IT
nza@cs.nott.ac.uk

Remedial classes

- Monday 3-5 in A32
- Contact Yan Su (yxs)
- Ordinary labs start on Tuesday

Plan of the lecture

- Classes and objects
- Constructors
- Creating objects
- Comparing objects
- Copying objects

Classes and Objects

- Classes are blueprints for objects
- They contain information on
  - which properties objects have (fields)
  - what they can do (methods)

Example: Point class

```
class  Point {
 int x,y;
 public Point(int x, int y){
    this.x = x;
    this.y = y;
 }
 public double distance(Point p){
    return Math.sqrt(Math.pow((p.x-x),2)
                + Math.pow((p.y-y),2));
// distance = √((p.x-x)² + (p.y-y)²)
 }}
```


Example: Person class

```
class  Person {
  String name;
  int age;
  Person(String name, int age){
    this.name = name;
    this.age = age;
  }
  void display() {
    System.out.println(name + ", " +
  age);}
}
```

Constructors

- Usually a class has at least one method which is used to create instances of a class (objects). It is called a constructor .
- Constructor has the same name as the class. A class can have several constructors with different sets of parameters.
- Usually they set all or some of the fields in the class to values passed as parameters.

Example: two constructors

```
class  Point {
 int x,y;
 public Point(int x, int y){
      this.x = x;
      this.y = y;
 }
 public Point () {
      this.x = this.y = 0;
 }
}
```


Example: default constructor

```
public Person(){
 }
```

(would set default values **`null`** for **`String`** name and 0 for **`int`** age).

Creating objects

```
Point p1;
```

Declares a variable of type Point. Does not allocate memory. Empty reference (refers to **`null`** object).

```
Point p1 = new Point(4,5);
```

Allocates memory (with "new"), executes the constructor. Now **`p1`** is a reference which points to the new object.

Before we go further… quiz:

```
Point p1;
System.out.println(p1.x);
// what will happen if you compile this?
Point p1 = null;
System.out.println(p1.x);
// what will happen if you compile this?
// what wil happen when you execute this?
```

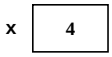

What happens when objects are constructed

Need to understand the difference between objects and basic types (boolean, char, byte, short, int, long, double, float).

Basic types

Basic type variables contain their value:

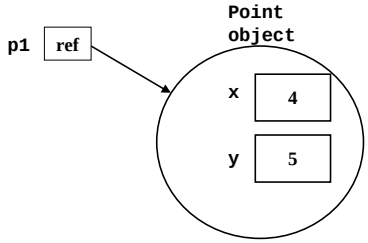
```
int x = 4
```



Reference types

- **`Person, Point, String etc.`** are reference type variables.
- The value of a reference type variable is a reference to (an address of) the value or set of values represented by the variable.

Creating objects

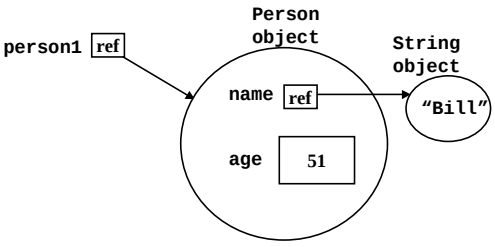


Example

```
Person person1 = new Person("Bill", 51);
```

Note that Strings are also objects (reference types).

Person object



Why does this matter?

- Reference type and basic type variables behave differently with respect to assignment and comparison.
- If you don't know that you will often be surprised at what your programs do.

Comparing objects

The following comparison:

```
person1 == person2
```

returns true if the references are to the same storage locations, and false otherwise. For example,

```
Person person1 = new Person("Bill", 51);
Person person2 = new Person("Bill", 51);
System.out.println(person1 == person2);
```

would print false , although the values of all fields in person1 and person2 are the same.

Equality

Suppose we want to find out whether object1 and object2 are equal (have the same values for all fields). Then we need to write a special method **`equals`** which compares objects of a given class. For example,

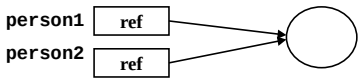
```
boolean equals (Person p) {
  return (this.name.equals(p.name) &&
          this.age == p.age);
}
```

Recall that String class has **`equals()`** method.

Assignment

```
person2 = person1;
```

results in person2 referring to the same locations as person1. If we change the values of person1's fields, the same changes will happen to person2 (because they refer to the same locations in memory).



Example

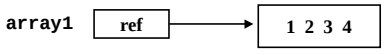
```
int[] array1 = {1,2,3,4};
int[] array2 = array1;
array1[0] = 0; // instead of 1
System.out.println(array2[0]);

// will print 0 although we "only"
// changed the other array!
```

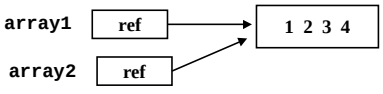
This is because arrays are reference types.

Example

```
int[] array1 = {1,2,3,4};
```

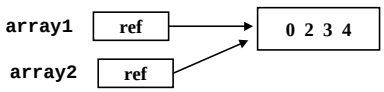



```
int[] array2 = array1;
```



Example

```
array1[0] = 0; // instead of 1
```



Making a separate copy

- Later in the course we will study cloning of objects.
- You can write a **`copy`** method yourself, for example:

```
public Person copy() {
  String newName = new String(this.name);
  String newAge = this.age;
  return new Person(newName, newAge);
}
```


To sum up

If x and y are variables of basic type,

- **`x = y`** means: "put the same value as y holds, in x's location"
- **`x == y`** means: "do x and y hold the same value?"

If x and y are variables of object / reference type

- **`x = y`** means: "get y to refer to the same location in memory as x"
- **`x == y`** means: "do x and y have the same address?"

Summary and further reading

- In this lecture, we recalled how to create objects and how objects are stored.
- It is important to understand that objects are stored as references, and how this affects comparison and assignment of objects. The reason objects are passed by reference, by the way, is that they can be very large and it is much more efficient to manipulate references than to copy around huge amounts of data.
- Reading: Java Gently Chapter 8 and Sun Java tutorial

http://java.sun.com/docs/books/tutorial/java/data/objects.html