<< Lecture 6 >>

Today's Plan

- Talk about homework 1
- A tiny little review of classes
- More on classes: the null keyword and member visibility
- Existing classes
  - The Math class as an example of static methods
  - The String class as an example of class as type
  - The Boolean, Integer, ..., classes, which are hybrid
- QUIZ 1 (it should not be hard at all)

Let us review through questions!

1. Can we compare the following pairs of types using ==?
   The == operation can compare (primitive) numbers or same types
   1. boolean == boolean **Yes**
   2. int == int **Yes**
   3. int == char **Yes**
   4. byte == boolean **No**
   5. long == byte **Yes**
   6. short == boolean **No**
   7. int == short **Yes**

2. Can we compare objects using:
   1. ==? **Same type**
   2. <, >, <=, >=? **No**

3. What is the value of **X** after the block? **X = 10**

```
int X = 0;
while( true ) {
      if ( X < 10 )
            X++;
      else
            break
}
```

4. What is the value of **X** after the block? **X = 0**

```
int X = 0;
for( X = 100 ; X > 0 ; X-- ) {
      if ( X < 0 ) break;
      System.out.println( X );
      continue;
}
```

5. Remove the useless instructions

```
int X = 0;
for( X = 1 ; X < 64 ; X = X * 2 ) {
      if (X > 0)   System.out.println( X );
            else   break;
      continue;
}
```

Ans.

```
int X;
for( X = 1 ; X < 64 ; X = X * 2 )
      System.out.println( X );
```

6. What is the value of **Y** after the block? **Y={'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'}**

```
char[] Y = new char[10];
for( int i = 0; i < Y.length ; i++ )
      Y[ i ] = 'a' + i ;
```

I am cheating here... I am going to talk about chars next class

7. Can static and non static members (variables, methods) coexist in the same class? **Yes**
8. Is it necessary to explicitly define a constructor for a class? **No**
9. What is **this**??? **The instance of the object at hand (the object itself)**
10. Is it possible to access **this** in a static context? **No**

LET US REVIEW "THIS"

- We used class Person last time

```
class Person {
  String name = "";
  int age  = 0;

  Person(String name, int age) {
      this.name = name;
      this.age  = age;
  }

  Person me() {
      return this;
  }

  void info() {
      System.out.println("Person "+name+" is "+age+" yo");
  }

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

- The first usage of this was for avoiding ambiguity—**this.name** versus **name**
- The second usage was for obtaining a *copy* of the instance itself; **this** is an object of type Person, and it is the one in use
- Let us try the following code to explore how Java sees things behind the scenes (*kind of*)

```
Person a, b, c;

a = new Person( "Zero" , 1000 );
b = new Person( "CopyX" , 10 );

c = a.me();

//Person doesn't have toString (!)
System.out.println( a );
System.out.println( b );
System.out.println( c );
```

The NULL keyword

- Do variables need to have a value assigned to them? No, unless they are basic (primitive) types
- A variable set to null does not have methods nor fields

```
Person p = null;
p.info()   // ERROR!!!

Person[] pp = null;
pp.length  // ERROR!!!
```

- An array can also be null!
- Often times, it is good to check that the variable is not null before using methods or fields

Visibility

- Variables and methods might not be visible (accessible) from other classes or packages
- Except for the **main**, we have not specified visibility in general; in such case, the members assume **default visibility**
- There are four levels of visibility:
  - **public**—everyone can see the member (method, field)
  - **protected**—visible to the package and the *subclasses*
  - **default** (unspecified)—everyone **in the package**
  - **private**—only the class itself can access the member
- The **protected** level will be interesting when we study inheritance in objected oriented design

- Since I have been coding simple classes, all accessing the same package, I have had no need to specify **public** members (except for the **void main**)
- Quick rule of thumb:
  - Make **public** all methods and fields that you want to access from outside the class
  - Make **private** anything that is too specific or algorithmic (i.e., related to the internals of the class code)
- We know the **static** modifier as well—anything left? **final**
- The **final** modifier is not for methods—is for variables only
- Any variable that is **final** is considered a constant—it cannot be changed (try to, and you will get an error)
- Why is useful to define variables with **final**? Very, very slightly faster code, basically

WELL FOLKS, THAT WAS ITERATION 1 !!!

It was rushed, but it covered enough basics to do lots already.

ITERATION 2 ???

- We will begin iteration 2 by reviewing what we have seen already, but adding slightly more detail to everything
  - Most details are necessary to know, but not really as practical (you can easily work around them)
- Then we will cover the object oriented paradigm—that's the point of iteration 2

- But let us start reviewing some basic classes

<< Lecture 7 will resume from here >>