CS101 | Ozyegin University

Java Standard Development Kit (SDK)

Throughout the semester we have been using the ACM Java Taskforce library in our programs. This library is not part of the official Java libraries as distributed by the owner of the language, Oracle, which became the owner by acquiring Sun Microsystems a few years ago.

When you write a program after you're done with CS101, you should prefer writing programs using official Java libraries. This is because, if you send someone (e.g. your IE102 or EE102 professor) a program that uses the ACM library, they will not be able to run your code since they most likely do not have the ACM library linked in their environment.

The official libraries of Java are called the Standard Development Kit (SDK).

We use the ACM library because it hides some of the clutter caused by Java's notation. This helps a beginner programmer to focus on the algorithmic details rather than language-wise details. Now that you've learned many things, we can use Java SDK in our programs without causing any mental shift.

Fortunately, writing console Java programs without the ACM library is easy; there are a few things you need to keep in mind. Writing graphical programs is a bit more involved. We will not talk about standard graphical library of Java; it is covered in CS102.

Writing console programs using JAVA SDK

The most basic program, **HelloWorld** is written as follows.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, SDK.");
    }
}
```

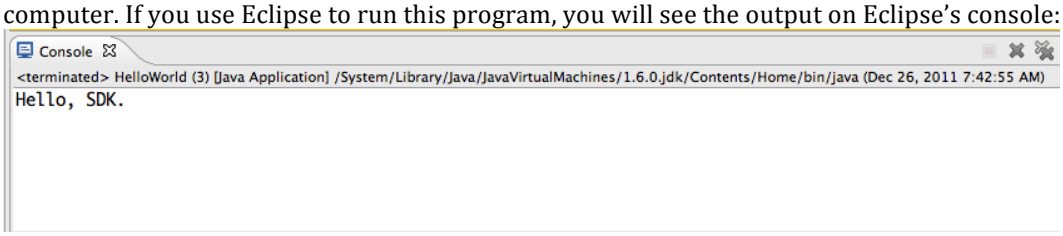
There are a few things to note in this program:

- We have not imported any packages. The **java.lang** package, which is imported by default, is sufficient.

- There is no **run** method; instead, there is the **main** method. Main is the entrance point to our program. The signature of the main method always have to be as

  ```
  public static void main(String[] args)
  ```

- To print a message to the screen, we use **System.out.println**. This means, we are sending the **println** message to **out** field of the **System** class. System.out refers to the standard output of our computer. If you use Eclipse to run this program, you will see the output on Eclipse's console:



Reading input from the user

This is slightly more involved. Let's take a look:

```
import java.util.Scanner;

public class InputOutput {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("I am a HAL 9000 computer.");
        System.out.println("I became operational at the H.A.L. plant in Urbana,
Illinois on the 12th of January 1992.");
        System.out.print("What's your name: ");
        String name = scanner.nextLine();
        System.out.println("Nice to meet you " + name + ".");
        System.out.print("In what year were you born? ");
        int year = scanner.nextInt();
        System.out.println("You are " + (2011-year) + " years old.");
        System.out.print("How tall are you, in meters? ");
        double heightInM = scanner.nextDouble();
        int heightInCM = (int)(heightInM * 100);
        System.out.println("You are " + heightInCM + " centimeters tall.");
        System.out.println("Bye!");
    }
}
```

To be able to read from the console, we first need to declare a **Scanner** object. **Scanner** is a class from the **java.util** package; hence, an import statement is needed. As the argument to the constructor of **Scanner**, we need to give the source of input. In this case, the source is the standard input (keyboard) of the computer, specified as **System.in**.

Once the scanner object is created, you can read input using the **nextLine()**, **nextInt()**, and **nextDouble()** methods. These methods correspond to the **readLine()**, **readInt()** and **readDouble()** methods in the ACM library, except that they do not take a String argument to print on the screen as the prompt message.

Methods

You can write methods the same way you have been doing, except for one thing: You need to define the methods to be **static**. In Java, non-static methods cannot be invoked directly from static methods. Because of this reason, and because **main** is a static method, you must make your methods static so that they can be invoked from inside **main**. E.g:

```
import java.util.Scanner;

public class Factorial {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter an integer: ");
        int n = scanner.nextInt();
        int factOfN = fact(n);
        System.out.println(n + "! = " + factOfN);
    }

    public static int fact(int n) {
        int product = 1;
        for(int i=1; i <= n; i++) {
            product *= i;
        }
        return product;
    }
}
```

The difference between static and non-static methods is covered in more detail in CS102.

Random Number Generation

To randomly generate numbers, you need to use the Random class from java.util package. Most useful methods of Random are **nextInt**, **nextDouble** and **setSeed**.

```
import java.util.Random;

public class Randomness {
    private static Random rgen = new Random();

    public static void main(String[] args) {
        int someInt = rgen.nextInt(10);
        System.out.println("A randomly generated integer in [0,10): " + someInt);
        double someDouble = rgen.nextDouble();
        System.out.println("A randomly generated double in [0.0, 1.0): " +
someDouble);
        rgen.setSeed(1);
        System.out.println("After setting the seed to 1: ");
        System.out.print(rgen.nextInt(100) + " " + rgen.nextInt(100) + " ");
        System.out.println(rgen.nextInt(100) + " " + rgen.nextInt(100));
        System.out.println("Setting the seed to 1 again, for verification: ");
        rgen.setSeed(1);
        System.out.print(rgen.nextInt(100) + " " + rgen.nextInt(100) + " ");
        System.out.println(rgen.nextInt(100) + " " + rgen.nextInt(100));
    }
}
```

Reading the contents of a file

```
import java.io.File;
import java.util.Scanner;

/* phonebook.txt must be placed to path workspace/CS101/
   Contents look similar to:

Lionel Messi 5559937
Erhan Erkut 5642000
Atom Karinca 9001234
Luke Skywalker 5675656
Peter Parker 1119999
Clark Kent 9879898
Sheldon Cooper 1235555
*/
public class FileInput {
    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(new File("phonebook.txt"));
        while(scanner.hasNext()) {
            String name = scanner.next(); // This is NOT nextLine()!!!
            String surname = scanner.next(); // This is NOT nextLine(), either!!!
            int phone = scanner.nextInt();
            System.out.println(name + " " + surname + " " + phone);
        }
        System.out.println("End of phonebook.");
    }
}
```

There are several things to note in the program above:

- When we created the scanner object, we specified **new File("phonebook.txt")** as the source of input.
- The input file must be placed under the folder workspace/CS101 (assuming that your project is called CS101)
- You may check whether the source still contains tokens, by sending the scanner **hasNext** message.
- Sending **nextLine()** message to the scanner object would return a whole line, up to the newline character. E.g. "`Lionel Messi 5559937`".
- Sending **next()** message to the scanner object will return the next token (i.e. word) E.g. "`Messi`".
- **main** method contains the following phrase: `throws Exception`
  This is required because we are trying to open a file and it's possible that the file does not exist, which would be an exceptional case. By writing "throws Exception" we are telling the Java compiler "Ok, I know that an exceptional case may occur, I'm aware of that, and I accept this as a fact of life." Exceptional cases can be handled using try-catch statements, which is a topic covered in CS102.