Handout 8
Defining Classes: method toString(), Constructors.

**public String toString()** – a special method Java calls to obtain a textual representation of an object. For example, when an object of that class is printed, or joined (+-ed) with a string. If a class does not define it, Java supplies a default version (via the process of inheritance).

If `toString()` is defined in a class, it is going to be used instead of the default one.

Example: let’s add the following method `toString()` of the Date class from the previous handout.

```java
/* * String toString() - returns a string with the description * of the calling object. That string would be printed * when a Date object is passed to System.out.println method */

public String toString(){
    return this.getMonth() + " " + this.getDay() + ", "+this.getYear();
}
```

Or, alternatively

```java
/** Methd toString() returns a String representation of the object. * That representation is used by system.out.println() and other * methods when they need to convert an object to a String */

public String toString(){
    return "Date object: " + this.month + " " +this.day + ", " + this.year;
}
```

If `toString()` method definition is not included in class Date, printing an object of class Date will display an internally created object id, like for example `Date@12340096`.

Once the `toString()` method is defined in class Date, printing an object of class Date will display a string that is returned by that method.

**Constructor:**
- Special kind of a method, has the same name as the class
- Are called when an object is created with `new`
- Purpose: initialize the instance variables of the object, when the object is created
- Multiple constructors within the same class may be provided for flexibility.
- If a class does not define a constructor – Java automatically creates a default constructor,
  - default constructor takes no parameters, (no-argument constructor)
  - initializes numeric instance variables to 0-s, all class type values to null
  - Example: `Date()` is a call to a default constructor in the following

  ```java
  Date date1 = new Date();
  ```

- **If any constructor is provided, then no constructors are supplied automatically**

Defining a constructor:
- Constructor methods have the same name as the class
- Have no return type definition in the header
Example – constructor for the class Date

Since the setter method is defined, call the set method here to take advantage of already defined code in method setDate() and the validation that it performs.

```java
/** Constructor initializes the fields of the object with parameters passed */
public Date (String mon, int day, int year ){
    //call method setDate to initialize the instance vars;
    boolean wasOK = this.setDate(mon,day, year);
    // if setDate indicates invalid arg...
    if (wasOK==false){
        System.out.println("Setting the date to Jan 1, 1970");
        this.setDate("Jan", 1, 1970);
    }
}
```

If Date class is augmented with either one of the above constructors, the following could be used to create a Date object:

```
Date date1 = new Date("Jan", 31, 2003);
```

BUT if now tried to use

```
Date date1 = new Date();
```

would get an error message from compiler (why?).

Overloading

Java allows to define methods with the same name and different signatures (i.e. parameter lists). Constructors can be overloaded. Consider another constructor, that does not take any arguments (a 0-argument, or default constructor)

```java
//define our own no-arg constructor
public Date (){  
    this.month = "Jan";  this.day = 1;  this.year = 2014;
}
```

Copy constructor: constructor that is passed an object, creates a copy of it.

Example:

```java
public Date(Date date){  
    // check if param date refers to object or is null
    if (date != null) {  
        this.setDate(date.month, date.day, date.year);
    }
}
Here’s a main method that uses different constructors of the Date class.

```java
public class DateDemo {
    public static void main(String[] args) {
        Date date1 = new Date("Apr", 11, 2007);
        System.out.println(date1); // method toString() is invoked within println
        Date date2 = new Date();
        System.out.println(date2); // method toString() is invoked within println
    }
}
```