Handout 9
Static variables and static methods.

Java variables:
- Local – declared inside a method, accessible only to that method.
  Formal parameters – like local variables, but declared in the parameter list, passed values when method is called.
- Instance – Declared within the class definition. When declared as private, accessible only from methods of their class, as part of an object. Denote the fields (a.k.a. attributes) of each object of the class, thus each object of the class has its own copy of each instance var.
- Static – declared within the class definition. There is a single copy of the static variable shared by all objects of the class in which the static variable is defined. Static variables can also be called ‘one-per-class’ variables. They are allocated even before any objects of the class are created.

Static variables are declared similarly to instance variables, except for the keyword static.

A public static variable (or constant, if declared with keyword final) can be accessed outside of its class using the class name prefix, e.g.
- Math.PI – static constant defined in Math class, denotes value 3.1415926
- Integer.MAX_VALUE constant defined in Integer class, denotes value 2147483647
- Employee.MIN_WAGE constant defined in Employee class, denotes value 8.5

A typical use for static variables is to count how many objects were created so far. Can be used to generate unique id numbers.

2. Static variables
- Declared with keyword static
- One per class (instead of one per object, as instance variable).
- Created independently of objects, outside of all objects.
- All objects (instances) of the class have access to it

Static constants
- Like static variables, except declared with keyword final
- If declared as public – can be used by other classes. Prefixed with the name of class in which they are defined, e.g.
- Math.PI or JOptionPane.QUESTION_MESSAGE

Example for which static variable is useful: counting number of objects, assigning unique ids
Suppose objects of `BankAccount` class need to have unique ids.

Practice – extend the following class definition to implement a unique id. Approach:
1. Add a new field (a.k.a. instance var) to store the id.
2. Assign value to id during object construction by using a counter of how many accounts were created so far. The use of the counter will guarantee uniqueness, because the counter will be incremented once a new object is created.

```java
public class BankAccount {

    private String owner;
    private double balance;

    private static int countAccounts = 0; // counter of all BankAccount objects that were created.

    // constructors
    public BankAccount (String owner){
        this.owner = owner;
        this.balance = 0;
        countAccounts++; //increase number of accounts created
    }

    public BankAccount (String owner, double balance){
        this.owner = owner;
        this.balance = balance;
        countAccounts++; //increase number of accounts created
    }

    // Accessor methods
    public String getowner(){    return this. owner;    }

    public double getbalance(){return  this.balance;   }

    // Method returns a string description of the bank account
    public String toString(){
        return  " Owner: " + this.getowner() +
                " Balance: " + this.getbalance();
    }

}
```
Static methods

Difference between the instance methods and static methods:

**Instance methods** implement object specific behaviors, in other words, they return value or perform an action that is based on the value of data attributes, i.e. instance variables of a concrete object. They are invoked with a calling object and are passed that object as an implicit parameter (this).

**Static methods**, on the other hand, do not require a calling object - they work independently of any object of their class, thus they cannot use any instance variables, because they are not passed a calling object. They are called with the name of the class, instead of the calling object.

**Instance method:**

```java
public static String monthName(int mNumber) {
    String[] monthNames = {
        "Nov", "Dec"};
    String mName = "UNKNOWN";
    if (mNumber>=1 && mNumber<=12)
        mName = monthNames[mNumber-1];
    return mName;
}
```

Using instance methods outside of class Date.

Notice the difference between the way a static method `monthName()` and the instance method `century()` are called.

```java
public class DateDemo{
    public static void main(String[] args) {
        Date date1 = new Date("Jan", 31, 2006);
        System.out.println (date1.century()); // instance method called
                                      // with a calling object
        System.out.println (Date.monthName(2)); // static method called
                                      // with class name
    }
}
```
Practice problem

1. Define static method isLeapYear() for the Date class. The method should be passed 1 integer, representing a year value and return a boolean reflecting to whether or not the year is a leap year. Only those years that are divisible by 4 are leap, except for those that are divisible by 100 and not divisible by 400.

   Show how the method should be called to determine if a year entered by user is a leap year.

2. Define a static method daysInMonth() for the Date class. The method should be passed two parameters: a month number and a year. The method should return an integer corresponding to the number of days in the specified month (taking the year into account for February) or -1, in case the month number is not valid.

   Show how the method should be called to determine the number of days in February of 2008.