More on Java (cont.)

Object-Oriented Programming
Outline

- Static methods
- Static variables (class variables)
- Packages

Readings:
- HFJ: Ch. 10.
- GT: Ch. 10.
Class methods

- Examples:

```java
int x = Math.round(42.2);
int y = Math.abs(-10);
```

- Methods in the Math class don't use any instance variable values. So they don't need to know about a specific Math object. All we need is the Math class.

- They were written as class methods – static methods.

- A class method (static method) is one that runs without any instance of the class.
Regular methods vs. static methods

Regular methods

```java
class Cow {
    String name;
    public String greeting() {
        return "Hi, I am " + name;
    }
}
```

- instance variable `name` affects the behavior of `greeting()`
- MUST be called using a reference variable:
  `s = cow1.greeting();`

Static methods

```java
class Math {
    public static int abs(int a) {
        if (a > 0) return a;
        return -a;
    }
    ...
}
```

- `abs()` has absolutely nothing to do with any `Math` instance variables
- CAN be called using the class name:
  `int a = Math.abs(-10);`
This won't compile

```java
class Duck {
    private int size;

    public static void main(String[] args) {
        System.out.println("Size of duck is " + size);
    }

    public void setSize(int s) {
        if (s > 0) size = s;
    }

    public int getSize() {
        return size;
    }
}
```

Which duck? Whose size?

If there's a duck or ten ducks on the heap somewhere, the static method doesn't know about any of them.

I've no idea which duck you are talking about!
Static method can't use instance variables or non-static methods

- Static methods can be called using class name
  - no this reference, no owner object

```java
public class Duck {
    private int size;

    public static void main(String[] args) {
        Duck d = new Duck();
        setSize(10);
        System.out.println("Size of duck is " + size);
    }

    public void setSize(int s) {...}

```
Static variables – class variables

- A class variable belong to the class, not any object.
- One copy shared among all class instances.

```java
public class Duck {
    private int size;
    public static int count = 0;

    public Duck() {
        count++;
    }
    ...
}
```

Each duck has its own size. But all ducks share the same count.
Static variables – class variables

```java
public class Duck {
    private int size;
    public static int count = 0;
    public Duck() {
        count++;
    }
    ...}
```

```java
class DuckTestDrive {
    public static void main(String[] args) {
        System.out.println(Duck.count);
        Duck d = new Duck();
        System.out.println(Duck.count);
    }
}
```

before any ducks are made

% java DuckTestDrive
0

after the first duck is created

1
Class/static variables

- belong to a class
- one copy shared among all instances of the class
- initialized before any objects of the class

Instance variables

- belong to an instance
- each instance has its own copy
- initialized when the owner object is created

```java
public class Duck {
    private int size = 0;
    public static int count = 0;
    
    public Duck() {
        count++;
        size++;
    }
}
```
Design pattern: Singleton

- Singleton: Ensure a class has only ONE instance, and provide a global point of access to it.

<table>
<thead>
<tr>
<th>Singleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>- static instance: Singleton</td>
</tr>
<tr>
<td>- Singleton()</td>
</tr>
<tr>
<td>+ getInstance(): Singleton</td>
</tr>
</tbody>
</table>

```java
if (instance == null) {
    instance = new Singleton();
}
return instance;
```

- Uses
  - In place of global variables
  - In system resource management
    - Avoid conflicting accesses from concurrent processes
Package: Declaration

- a **package** statement appears as the first non-comment in the file

```java
// HelloMsg.java
package hanv;

public class HelloMsg {
    public void sayHello() {
        System.out.println("Hello, world!");
    }
}
```

Declared as **public** so that they can be used outside package **hanv**

package declaration with package name. The rest of the file belongs to the same package
Package: Usage

- Two ways:

1. Use the `import` statement to make the name(s) in the package available, once for all.

```java
//Hello.java
import hanv.HelloMsg;

public class Hello {
    public static void main(String[] args) {
        HelloMsg msg = new HelloMsg();
        msg.sayHello();
    }
}
```

2. Give the fully qualified name at every call.

```java
//Hello.java
public class Hello {
    public static void main(String[] args) {
        hanv.HelloMsg msg = new hanv.HelloMsg();
        msg.sayHello();
    }
}
```
Package – Compile and run

- Compile
  javac HelloMsg.java -d <class_root_dir>
  javac Hello.java

- Run
  java Hello
Package – make it simple

- Where to put source files?
  - C:\java root directory
  - C:\java\hanv classes in hanv package

- Compile: stay at the root!
  - C:\java\> javac hanv\HelloMsg.java
  - equivalent to javac hanv\HelloMsg.java –d .
  - or javac hanv\HelloMsg.java –d c:\java

  - C:\java\> javac Hello.java

- Run
  - C:\java\> java Hello