Objects and Classes

Object-Oriented Programming
Outline

- Classes vs. objects
- Designing a class
- Methods and instance variables
- Encapsulation & information hiding

Readings:
- HFJ: Ch. 2, 3, 4.
- GT: Ch. 3, 4.
A Java program, at run-time, is a collection of objects. They do things (their methods) and ask other objects to do things (calling methods of others).

A Java program, when we write it, is a collection of classes

A Java library contains predefined classes that we can use in our programs
Classes vs. objects

- A class is a **blueprint/template** that is used to construct objects.
- Each object is **instantiated** from a class. That object is called an **instance** of the class.
Designing a class

- When you design a class, think about the objects that will be created from that class
  - things the object **knows** about itself
  - things the object **does**
Designing a class

- things the object knows about itself
  - **instance variables**
    - the object's instance variables represent its *state*
- things the object can do
  - **methods**
    - the object's methods represent its *behavior*
Writing a class

1. Write the class

class Dog {
    int size;
    String breed;
    String name;

    void bark() {
        System.out.println("Ruff! Ruff!"};
    }
}

instance variables

a method

DOG
size
breed
name
bark()
Writing a class

2. Write a tester (TestDrive) class with code to test the Dog class

```java
public class DogTestDrive {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.name = "Bruno";
        d.bark();
    }
}
```

*Information hiding is not here yet.*
Writing a class

Instance variables/methods belong to an object. Thus, when accessing them, you MUST specify **which object** they belong to.

```java
public class DogTestDrive {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.name = "Bruno";
        d.bark();
    }
}
```

- **dot notation (.)** and **the object reference**
- access 'name' of the **Dog**
- call its *bark*() method
Object references

3 steps of object declaration, creation and assignment:

1. Declare a reference variable
   ```java
   Dog myDog = new Dog();
   ```

2. Create an object
   ```java
   Dog myDog = new Dog();
   ```

3. Link the object and the reference
   ```java
   Dog myDog = new Dog();
   ```
Object references

Dog myDog = new Dog();

Remember: References are not objects!
Messaging between objects

- Sending a message to an object is actually calling a method of the object.
  
  ```
  d.bark()
  ```

- Syntax:
  ```
  <object reference>.<method_name>(<arguments>)
  ```

  - recipient
  - message content
  - extra information
Methods – How objects behave

- Objects have
  - state (instance variables)
  - behavior (methods)
- A method can use instance variables' value and change the object's state.
- A method can use instance variables so that objects of the same type can behave differently.
State affects behavior, behavior affects state

class Dog {
    int size;
    String breed;
    String name;

    void bark() {
        if (size > 14)
            System.out.println("Ruff! Ruff!");
        else
            System.out.println("Yip! Yip!");
    }

    void getBigger() {
        size += 5;
    }
}

State affects behavior. Dogs of different sizes behave differently.

method changes state

<table>
<thead>
<tr>
<th>DOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
</tr>
<tr>
<td>breed</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>bark()</td>
</tr>
<tr>
<td>getBigger()</td>
</tr>
</tbody>
</table>
State affects behavior, behavior affects state

class DogTestDrive {

    public static void main (String[] args) {

        Dog one = new Dog();
        one.size = 7;
        Dog two = new Dog();
        two.size = 13;

        two.bark();
        two.getBigger();
        two.bark ();

        one.bark();
    }
}

Đại học Công nghệ - ĐHQG HN  
Objects and Classes  
15
State affects behavior, behavior affects state

class DogTestDrive {
    public static void main (String[] args) {
        Dog one = new Dog();
        one.size = 7;
        Dog two = new Dog();
        two.size = 13;
        two.bark();
        two.getBigger();
        two.bark();
        one.bark();
    }
}
State affects behavior, behavior affects state

class DogTestDrive {

    public static void main (String[] args) {

        Dog one = new Dog();
        one.size = 7;
        Dog two = new Dog();
        two.size = 13;

        two.bark();
        two.getBigger();
        two.bark();
        one.bark();
    }
}

%> java DogTestDrive

Dog object 1
  name: null
  size: 7
  breed: null
State affects behavior, behavior affects state

```java
class DogTestDrive {
    public static void main (String[] args) {
        Dog one = new Dog();
        one.size = 7;
        Dog two = new Dog();
        two.size = 13;

        two.bark();
        two.getBigger();
        two.bark();
        one.bark();
    }
}

%> java DogTestDrive
```

State affects behavior, behavior affects state

class DogTestDrive {
    public static void main (String[] args) {
        Dog one = new Dog();
        one.size = 7;
        Dog two = new Dog();
        two.size = 13;

        two.bark();
        two.getBigger();
        two.bark();
        one.bark();
    }
}

%> java DogTestDrive
Yip! Yip!
State affects behavior, behavior affects state

class DogTestDrive {
    public static void main (String[] args) {
        Dog one = new Dog();
        one.size = 7;
        Dog two = new Dog();
        two.size = 13;
        two.bark();
        two.getBigger();
        two.bark();
        one.bark();
    }
}

%> java DogTestDrive
Yip! Yip!
State affects behavior, behavior affects state

class DogTestDrive {
    public static void main (String[] args) {
        Dog one = new Dog();
        one.size = 7;
        Dog two = new Dog();
        two.size = 13;

        two.bark();
        two.getBigger();
        two.bark();

        one.bark();
    }
}

%> java DogTestDrive
Yip! Yip!
Ruff! Ruff!
State affects behavior, behavior affects state

class DogTestDrive {
    public static void main (String[] args) {
        Dog one = new Dog();
        one.size = 7;
        Dog two = new Dog();
        two.size = 13;
        two.bark();
        two.getBigger();
        two.bark();
        one.bark();
    }
}

$> $ java DogTestDrive
Yip! Yip!
Ruff! Ruff!
Yip! Yip!
$>
Instance variables vs. local variables

Instance variables
- belong to an object
- declared inside a class but NOT within a method
- have default values (0, 0.0, false, null...)

```java
class Dog {
    int size;
    String breed;
    ...
    void getBigger() {
        size += 5;
    }
}
```

Local variables
- belong to a method
- declared within a method
- MUST be initialized before use

```java
public class DogTestDrive {
    public static void main(String []
        Dog dog = new Dog();
        dog.name = "Bruno";
        dog.bark();
    }
}
```
Encapsulation / information hiding

- What is wrong with this code?
  - It allows for a supernatural dog
  - Object's data is exposed.

- Exposed instance variables can lead to invalid states of object

- What to do about it?
  - write set methods (setters) for instance variables
  - hide the instance variables to force other code to use the set methods instead of accessing them directly.
Information hiding. Rule of thumb

- Mark instance variables **private**.
- Make getters and setters and mark them **public**.

- Don't forget to check data validity in setters.

```java
class Dog {
    private int size;

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

    public int getSize() {
        return size;
    }

    ...
```
Class access control

Access modifiers:
- public : Accessible anywhere by anyone
- private : Only accessible within the current class
- protected : Accessible only to the class itself and to its subclasses or other classes in the same "package"
- default (no keyword): accessible within the current package
Implementation vs. Interface

- **DogTestDrive**: a “client” of Dog
- **Implementation**
  - Is the data structures and code that implement the object features (instant variables and methods)
  - Usually more involved and may have complex inner workings
  - Clients don’t need to know
- **Interface**
  - The controls exposed to the “client” by the implementation
  - The knobs on the black box
Encapsulation / information hiding

“Don’t expose internal data structures!”

- Objects hold data and code
  - Neither is exposed to the end user or "client" modules.
- Interface vs. implementation
  - A cat's look vs. its internal organs
  - A TV's screen & buttons vs. the stuff inside the box
- Complexity is hidden inside the object
  - Make life easier for clients
  - More modular approach
    - Implementation changes in one component doesn’t affect others
  - Less error-prone