CS100M
Introduction to Computer Programming

Spring 2004
Lectures 21-22
OOP & References

Announcements
- A5 due
- A6 is last assignment
- Prelim 3: Tuesday
  - Topics include all Java up to and including this lecture
  - Review material, room, review session in Prelims (review session will be Sunday)
  - Structure of exam

Summary/Overview
- Problem solving with OOP
  - Nouns: class, field, local var, constant
  - Verbs: method, operator
- Encapsulation:
  - Has-a relationship
  - Information hiding
- Mechanics of OOP
  - References and objects
  - Pass by Value
  - this
  - aliases

Reference
- Rem:
  - Operator new returns reference to a newly created object
    new Thing()
  - Reference variable is a variable that stores the reference to an object:
    Thing t = new Thing();
- Questions:
  - Does t store the object?
    eg)
  - What is a reference?
    eg)
  - Can you directly use references?
    eg)
Rough Memory Model

- The Stack: Methods and local variables:
- The Heap: Objects and their fields
- Reference: address

Reference Values

- Java does not allow explicit changes:
  ```java
  Thing t = new Thing();
  t = 1027; // crap!
  ```
- Use `toString` to help see references:
  - If you do not provide a `toString` method, Java gives you a default version
  - Default version usually returns a String version of the object's address in memory
  - Not really useful except when learning about references

Example

```java
public class References {
    public static void main(String[] args) {
        Thing1 t1 = new Thing1();
        System.out.println(t1);
        System.out.println(new Thing1());
        System.out.println(new Thing2());
    }
}

class Thing1 { }

class Thing2 {
    public String toString() {
        return "hello";
    }
}
```

Special References

- `null`
  - Placeholder for “no object”
  - Effectively, a “zero address”
  - Why bother? Think of variable rules...
- `this`
  - Means, “the current object”
  - Two places to use:
    - As a reference to the current object to access the object's fields or methods without worrying about scope
    - As a way to call another constructor from a constructor
Example

```java
public class SpecialRefs {
    public static void main(String[] args) {
        Person p1, p2; // current value?
        p1 = new Person("Dimmu", null);
        p2 = new Person("Borgir", null);
        p1.setFriend(p2.getMe());
        p2.setFriend(p1.getMe());
        System.out.println(p1);
        System.out.println(p2);
    }
}
```

class Person {
    private String me; // value?
    private String friend; // value?
    public Person(String me, String friend) {
        this.me = me;
        this.friend = friend;
    }
    public void setFriend(String friend) {
        this.friend = friend;
    }
    public String getMe() {
        return me;
    }
    public String toString() {
        return "I am \"" + me + ", and my friend is \"" + friend + \"\.",\n    }
}

Aliases

- Can't change reference values but you can “pass” them!
- Example
  ```java
  Thing t1, t2;
  t1 = new Thing();
  t2 = t1;
  t2.changeSomething();
  // what happens to t2? object?
  ```
- Alias: variable that refers to the same object as another variable
  - References help to connect data together (data structures)
  - Alias provides mechanism to move “pointer” in data
  - Alias also way of swapping (min, max, ...)
  - Helps to allow methods to change data “inside” an object

Example

```java
public class Aliases2 {
    public static void main(String[] args) {
        Book b1 = new Book("Stand on Zanzibar");
        System.out.println(b1);
        Book b2 = b1;
        b2.pages = 100;
        System.out.println(b1.pages);
    }
}
```

class Book {
    public int pages;
    public String name;
    public Book(String name) {
        this.name = name;
    }
    public String toString() {
        return "I am \"" + name + \"\.",\n    }
}

Pass By Value

- Reminder: all methods pass by value
  - Parameter values are copied from actual arguments to formal parameters
  - No way in Java to pass an “entire” variable
- What if scenario?
  - Create an object and store in a var
  - “Pass the var” (actually, just the val) to another method
  - What happens to the variable? Object? Fields?
- Example:
  - See next page...
  - Then see the page that follows....
Motivating Example

```java
public class Pass1 {
    public static void main(String[] args) {
        Person p = new Person();
        p.name = "Dimmu";
        change(p);
        System.out.println(p);
    }
    public static void change(Person p) {
        p.name = "Borgir";
        p = null;
    }
}
class Person {
    public String name;
    public String toString() { return name; }
}
```

What is happening?!?

- Recall these rules:
  - Variables store values
  - Reference variables store object addresses, which must also be values
  - Java methods pass values to input parameters
  - Scope of variables: look at current block; not found? See enclosing block (and so forth)
    - Method parameters and local variables never seen outside method
    - Only variables seen outside of method are fields (need to use `this` if field and method name the same)
  - Dot operator used in syntax `var.member` to access `member` of object that `var` refers to
  - Alias: ref that has the same address as another ref

Putting it together

- You cannot change an object by resetting a variable in another method!
- But you can “get inside” an object and change its fields and access its members because an aliased variable will share the same object!
- Now go back and review previous example
- Another example....

Example

```java
public class Aliases {
    public static void main(String[] args) {
        Person p1 = new Person("Dimmu");
        Person p2 = new Person("Borgir");
        p1.makeFriends(p2);
        System.out.println(p1);
        System.out.println(p2);
    }
}
```
Example Continued

```java
class Person {
    private String name;
    private Person friend;
    public Person(String name) {
        this.name = name;
    }
    public void setFriend(Person friend) {
        this.friend = friend;
    }
    public void makeFriends(Person friend) {
        friend.friend = this;
        this.friend = friend;
    }
    public String toString() {
        return "I am \"+name+\", and my friend's name is \"+friend.name+\".";
    }
}
```

Static

- The gist:
  - Sometimes you want a mechanism for accessing members without creating an object
  - Modify a member (nothing else!!!) with `static` modifier
  - So, static fields will be shared by all objects of the same class!
- Syntax for accessing a static member:
  `Classname.member`  
- You can also use standard OOP techniques to create objects and access members

Syntax Example

```java
public class StaticTest {
    public static void main(String[] args) {
        Person.name = "Zardoz";
        Person p = new Person();
        p.name = "John";
        Person q = new Person();
        System.out.println(p);
        System.out.println(q);
    }
}
```

More Practical Example

```java
class Person {
    private String name;
    private static int count;
    public Person(String name) {
        this.name = name;
        count++;
    }
    public static int getCount() { return count; }
}
```

```java
class Student {
    private String name;
    private static int count;
    public Student(String name) {
        this.name = name;
        count++;
    }
    public static int getCount() { return count; }
}
```

```java
class StaticTest2 {
    public static void main(String[] args) {
        System.out.println(Student.GRADYEAR);
        Student s1 = new Student("Dani");
        Student s2 = new Student("Shagrath");
        Student.currentYear = 2001;
        System.out.println(s2.currentYear);
        System.out.println(Student.getCount());
    }
}
```