Lecture 11: Constructors in Java

Last time:
1. Object equality
2. Objects and classes in Java
3. Methods

Today:
1. Project #2 Hints
2. Aliasing
3. Constructors, Accessors, Mutators
4. Equality
5. Printing an object
6. for loops
Project #2 Is Assigned!

- The assignment is on the CMSC 131 web-site (click “Projects” link).
- It is due Saturday, 2/24 at 11 pm
- The project is open
- Start now!
  - Read entire assignment from beginning to end before starting to code
  - Check out assignment now from CVS
  - Follow the instructions *exactly*, as much of grading is automated
Project Hints

- In Project #2: implement a method that
  - Takes as input
    - A grid object
    - Letters 'L', 'O', 'J', 'T', 'H' or 'X'
    - Color
  - Prints given letter on grid using given color
Example: ‘L’

9x9 grid

17x17 grid
Another Example: ‘O’
Grids? Colors?

- You don’t have to implement these
- The assignment explains how to import them, what operations you need
- Your main task: figure out how to draw letters by filling in squared on grid
How To Do This?

One approach:
- Treat each letter as a special case
- Tedious, error-prone

Better:
- Identify common operations in letters
- Implement methods for these operations
- Call these methods in the letter-drawing method
- Example
  - ‘L’, ‘O’ both use horizontal, vertical strokes
  - Can you implement methods for these strokes?
Aliasing

- Recall Student class definition from last time
  ```java
  public class Student {
    public String name;
    public int id;
    public int tokenLevel;

    public void sayHello () ...
  }
  ```

- What is printed as result of following?
  ```java
  Student s1 = new Student();
  Student s2 = s1;
  s1.id = 123456789;
  System.out.println(s2.getLastFourDigits());
  ```
  - 6789
Why? Aliasing!

- $s_1$ is a reference variable (value is an address)
- $s_2$ is a reference variable
- Address stored in $s_1$ is same as address stored in $s_2$
  - $s_1$ and $s_2$ refer to the same object (are aliased)
  - So changes to $s_1$ affect $s_2$, and vice versa
Aliasing Example

```java
Student s1 = new Student();
Student s2 = s2;
s1.id = 123456789;
System.out.println(s2.getLastFourDigits());
```

- `6789` is printed
Be Careful about Aliasing!

- Subtle errors
- Note: `==` can be used to check for aliases
  If a, b are reference variables then `a == b` holds if and only if a, b are aliased
Constructors

- Special “methods” in class definitions to specify how objects are created
- Form of a constructor definition:
  
  ```
  Student (String nameDesired, int IDDesired, int tokensDesired) {
      name = nameDesired;
      id = IDDesired;
      tokenLevel = tokensDesired;
  }
  ```

  - Can have more than one constructor, provided argument lists are different
  ```
  Student (int IDDesired) {
      id = IDDesired;
  }
  ```

  - Java includes default constructor (no arguments), which you can redefine (override)
  ```
  Student () {
      tokenLevel = 3;
  }
  ```
Set / Get Methods

- We have been using = to modify instance variables and accessing variables directly to read values.
- Generally, this is not good practice because it imposes restrictions on class implementation.
- Better
  - set methods to set values (mutators)
  - get methods to read values (accessors)
Set Methods (Mutators)

```java
public void setID (int newID) {
    id = newID;
}

• Can also do consistency checking

public void setTokenLevel (int newTokenLevel) {
    if (newTokenLevel <= 3) {
        tokenLevel = newMonth;
    } else {
        System.out.println ("Bad argument to setTokenLevel: " + newTokenLevel);
    }
```
Get Methods (Accessors)

- Sole purpose is to return values of state
  
  ```java
  public int getID () {
      return id;
  }
  ```

- Why use them?
  - The state information may not always be stored in a single instance variable, since implementations may change
  - You give designers option of changing instance variables
Equality Testing

```java
public boolean equals(Student otherStudent) {
    return id == otherStudent.id;
}
```
Objects to Strings

- What happens if we try to print a Student object?
  - invoke `println` using a `Student` object as an argument?
    ```java
    Student s1 = new Student ();
    System.out.println (s1);
    ```
- Something like this prints:
  ```java
  Student@82ba41
  ```
- ???
Java Knows “How” To Print Any Object

• Why?
  • Every class has a default `toString` method
  • `toString` converts objects into strings
  • `System.out.println` calls this method to print an object
  • Default: object type and address

• `toString` can be overridden!

```java
// The method for converting Students to strings

public String toString () {
    return (name + ": " + id);
}
```
for Loops

- Three kinds of loops in Java
  - while
  - do ... while
  - for

- A common programming idiom
  ```java
  int i=0;
  while (i <= 10) {
      j += i;
      i++;
  }
  ```

- Equivalent for-loop
  ```java
  for (int i=0; i<=10; i++)
      j += i;
  ```
for -Loop Form

for (<init>; <continue>; <increment>) <stmt>

- Equivalent to:
  <init>;
  while (<continue>) {
    <stmt>;
    <increment>;
  }

- Any of the three <init> / <continue> / <increment> components may be omitted
  for (; ;) {
    ...
  }

Runs forever!