CMSC 131
Object-Oriented Programming I

Classes Introduction V

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane

Overview

- Private vs. Public
- Equals method
- toString method
- Static variables

Some Sites

- First bug
  - http://www.jamesshuggins.com/h/tek1/first_computer_bug.htm
- To keep informed
  - http://slashdot.org/

Private vs. Public

- What is the effect of private?
- What is the effect of public?
- Let's define a class and see how access is affected when defining methods and fields using private and public
- Why we want to declare fields as private?
  - Data protection
- Why we want to declare methods as private?
- By having fields define as private we need to provide get/set methods
  - Let's add get/set methods to our previous class
  - Eclipse allows you to generate set and get methods
- What if we don't provide an access specifier?

Equality Testing

```
public boolean equals(Student otherStudent) {
    if (otherStudent == null) {
        return false;
    } else if (id == otherStudent.id) {
        return true;
    } else {
        return false;
    }
}
```

- IMPORTANT: For now we will have a parameter different from Object but the correct approach to define equals is to use Object as a parameter
- Let's add an equals method to our previous class

Objects to Strings

- What happens if we try to print a Student object?
  - Invoke println using a Student object as an argument?

```
Student s1 = new Student ();
System.out.println (s1);
```

- Something like this prints:

```
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!

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

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

- Let's add a toString method to our previous example

Initialization of Variables

- Remember than instance variables have default values if they are not initialized
  - boolean → false
  - numeric → zero
  - references → null
- Instance variables can be initialized when they are declared
- Instance variables can be initialized in the constructor
- The constructor will override

Static Variables

- We have seen static methods
  - Methods that do not require an object in order to be called
  - They don't refer to any instance variables
  - We called them using the name of the class
  - They can also be called using an object of the class but that is not recommended (e.g., you get warning in Eclipse)

Static Variables

- We can have static variables
  - Variable that is shared by all instances of the class
  - There is only one copy
  - You can access it by using the name of the class

    ClassName.*staticVariableName*
  - In methods of the class you can access the variable directly
    - No need for class names
- Let's see a diagram that shows how static variables are shared
- Let's add a static variable to our previous example
- When is a static variable initialized?
  - At class load time

Scope

- What is the scope (visibility) of:
  - Instance variables
  - Static variables
  - Local variables
  - Parameters

Review Of Variables

- **Instance variables**
  - Belong to the class and created when an object is created
  - Space for them exists in the heap
  - We need an object to access them
  - In a non-static method we can refer to them directly (no need for object reference)
  - They have default values
- **Static variables**
  - Belong to the class
  - They are shared by all instances of the class
  - We can refer to them by using the class name (no need for class name if reference by methods of the class)

Review Of Variables

- **Local Variables**
  - Defined in a method
  - Created when method is called and destroy on exit
  - Don't have default values (Eclipse will know if you are using uninitialized variables and force you to initialized them)
- **Parameters**
  - Like local variables but initialized when a method is called