More on Classes

Chris La Pilla
CompSci 6
Summer 2007

Static variables

• We’ve seen that each object contains its own copies of instance variables
• What if we want a variable to store information about the entire class of objects, instead of about a particular object?
• E.g. how can we keep track of the total number of StudentRecords we’ve created so far, so that we can assign each student a unique ID?

Static variables

• A static variable stores a value that is not associated with any particular object:
  
  private static int numStudents = 0;

• Like instance variables, it has an access specifier and a type
• However, there is only ONE copy of the variable, regardless of how many objects we create

Accessing static variables

• Static variables are used just like instance variables from methods within the class
• Recall that we access public instance variables by using the dot operator on an object reference, e.g.
  
  Dimension size = new Dimension(10, 20);
  System.out.println(size.height);

• For public static variables, we use the class name instead, e.g.
  
  System.out.println(StudentRecord.numStudents);

Static methods

• A static method is a method that is not invoked on any object
• Declared with the static keyword; e.g. in the Math class:
  
  public static double sqrt(double x)

• Call a static method on the class name, just as with public static variables:
  
  double ans = Math.sqrt(2.0);

• Static methods are frequently used to perform computations on primitive types (why?)

Examples of static variables and methods

• Variables:
  - Constants in Math class: Math.PI, Math.E
  - System.out / System.in
  - Integer.MAX_VALUE, Integer.MIN_VALUE

• Methods:
  - All methods in Math class
  - Integer.parseInt(String s): converts s to an int
  - All main methods!
Choosing classes

- Classes should represent a single entity or concept
- A good rule of thumb is that the class name should be a noun
- There are some special uses of classes:
  - *Actor classes*: classes whose primary job is to do something, not to hold information (e.g. Scanner, Random)
  - *Utility classes*: classes that hold a collection of static methods – they have no objects! (e.g. Math)
  - Some classes only contain a main method

Types of methods

- *Accessor method*: returns information about the object (e.g. getName(), computeGrade())
- *Mutator method*: changes the state of an object (e.g. addGrade())
- Classes that have no mutator methods are called **immutable** (e.g. String)

Method parameters

- Parameters supplied in the parentheses of a method call are *explicit parameters* – They are copied into the parameter variables
- Reference to the object on which the method is called is an *implicit parameter* – The reference to the object is stored in the special variable `this`
- Static classes have no implicit parameter
- Example: `record.addGrade(85)`;
  - `85` is an explicit parameter
  - `record` is the implicit parameter

Explicit Parameters

- *Primitive types*: changes to parameter variables holding primitive types do not affect value of variables in the calling code (why?)
- *Object references*: changes made to state of objects referred to by parameter variables can affect other parts of the program (why?) – These changes are called side effects
  - You should generally avoid side effects unless the purpose of the method is to modify object parameters
  - Method documentation MUST state side effects

Preconditions and Postconditions

- A *precondition* is a requirement that the caller of a method must meet
- A *postcondition* is a condition on the return value or state of the object that will be met if the preconditions are met
- Can be thought of as terms of a contract between the method and the caller:
  - If the preconditions are met when the method is called, the method guarantees the postconditions will be met
  - Otherwise, the method is not responsible for what happens

Example: Student Record

- What preconditions should the addGrade() method in the StudentRecord class have?
- What preconditions should the computeAverage() method have?
- What about postconditions?
Variable scope

- The scope of a variable is the part of the program in which the variable can be accessed
- A local variable is a variable declared inside a method
- Scope of a local variable extends to the end of the block { } in which the variable was declared

Example

```java
if (num > 5){
    String result = "Greater than 5.";
} else{
    String result = "Less than or equal to 5.";
}
System.out.println(result);
```

- Syntax error on last line! (why?)

Example

```java
for(int i = 0; i < 10; i++){
   ...
}
System.out.println(i); //syntax error!
```

```java
int i = 0;
for (i = 0; i < 10; i++){
   ...
}
System.out.println(i); //OK
```

Variable Scope

- We've already seen that a local variable can have the same name as an instance variable
- This is called shadowing
- Can access the class variable by using the `this` reference