CS133: Developing Programming Principles

Lecture 5

Objects, Classes, Methods, Constructors, Local and Instance Variables

Object

- A Java program consists of interacting objects.
- An **object** models a thing (real or abstract). It has values (attributes) and operations (methods).
- A **class** in Java is the code which details how to create (instantiate) an object of the class and includes the set of associated operations. A "template".
- An object is an instance of a class

Objects and Classes

- Recall: A **class** defines **data** (attributes) and **operations** on that data (methods).
  - Thus, a class defines a type!
- An **object** is space in memory for the data specified by a class.
  - We say: an object **instantiates** a class.
  - A Java program consists of interacting objects.
- Classes define types that **model** things:
  - Abstract things: coordinates, rectangles, fractions...
  - Real things: students, boards, cash registers...

Example of an object

- We can model a cash register.
  - Data: balance, number of sales, number of items sold, amount of revenue
  - Operations: check in an item, calculate subtotals, receive payment...
- This defines a cash register "type".
- We can make the model simpler or more complex.

Instantiating an object

- Syntax:
  Class_Type Name **= new** *Class_Type***();**

  Examples:

```
Scanner input =
       new Scanner(System.in);
CashRegister till =
       new CashRegister();
```


Structure of a Java class

```
public class ClassName
{
  // declare instance variables
  // declare methods
}
```

Instance variables

- Store information about an object.
- Declared in a class but outside a method.
- Accessible to all methods in the class.
- Each object of a class has its own values for the instance variables.

Declaring instance variables (primitives)

- Syntax (for primitives):

  `private` *Type varName*`;`

  or

  `private` *Type varName* **`=`** *value*`;`

  Examples:

```
private int numSales = 0;
private double balance = 100.00;
```

Declaring instance variables (objects)

- Syntax:

  `private` *Class_Type varName* **`=`**

  `new` *Class_Type*`();`

  Or

  `private` ***`Class_Type varName`***`;`

Methods

- A method is a definition of some action that the object can perform.

- Examples:

  `String`: **`toLowerCase`**, **`length`**

  `Scanner`: **`nextLine`**, **`nextInt`**

  `Board`: **`getPosition`**, **`putPeg`**

Our object's methods

Cash register objects can:

- Scan an item to be purchased
- Return a subtotal
- Receive payment for a sale ("complete" the sale)
  - Maybe make change?
- Indicate the number of sales made
- Indicate the number of items sold
- Indicate the amount of revenue earned
- Receive cash for their balance
- ...

Types of methods

- **`void` methods**

  perform some action but don't return a value.
  Often change values of instance variables.

  Signature:
  `public void` *methodName*`(`*Parameters*`)`

  Example: **`close()`** from the **`Scanner`** class.

Types of methods continued

- **Methods that return a single value**
  should only perform the actions necessary to return the value. Answers a question.

  Signature:
  `public` *Return_Type methodName*(*Parameters*)

  Examples:

`public int length()`	**`String`** class
`public double nextDouble()`	**`Scanner`** class
`public String toLowerCase()`	**`String`** class

Partial implementation

```
public class CashRegister
{ private double balance = 0.00;
  private int numSales = 0;
  private double currentTotal = 0.00;
  // other instance variables omitted

  public void addToBalance(double amount) {…}
  public void scan(int quantity,
     double unitPrice, String description) {…}
  public int getNumSales() {…}
  // other methods omitted
}
```

Types of methods: Constructors

- **Constructors**

  Constructors are special methods, which are only called when an object is created using **new**. Generally, they initialize the object. They have the same name as the class and **no** return type.

  Signature:

  `public` *className***`(`***Parameters***`)`**

  Examples:

  `public Scanner(InputStream source)`

Constructors

A class may have several constructors:

```
public CashRegister(
        double initialBalance);
public CashRegister();
```

Body of a constructor

- Includes assignments to instance variables.
- Parameters can be used in the assignment statements.
- Should include any other actions to be taken when an object is constructed.

Default constructors

- If you don't specify a constructor in the class, Java automatically adds an empty one that does nothing but create a memory reference and return it.
- Instance variables must be initialized when declared, or by using mutator methods.
- If at least one constructor is defined, Java does not create a default constructor.

Use default constructor or write one?

- You almost always want to write constructors.
- Instance variables should be initialized appropriately before object is used. How?
  - Default constructor, combined with mutator methods (user's responsibility). (Often bad practice)
  - Specialized constructor (programmer's responsibility).
- If "other actions" are required, specialized constructor must be written.

Invoking a method

- **`void` methods**

  objectName`.`*method*`(` *Arguments*`);`

  Examples:

  `theTill.addToBalance(5.00);`

  (where **`theTill`** is of type **`CashRegister`** and has been initialized)

Invoking a method continued

- **Methods that return values**

  Variable **=**

  objectName`.`*method*`(`*Arguments*`);`

  Examples:

```
int numChars = myString.length();
double amountToPay =
      theTill.getSubtotal();
```


Writing a method

- Methods are part of a class

```
Method_Signature
{
 // Method body:
 // Java code to solve the problem
 // or answer the question.
}
```

Writing a method: **`addToBalance`**

```
public class CashRegister
{
  private double balance;

  public void addToBalance(double amount)
  {
      // Increment the balance in the register.
      // This amount had better be non-negative!
      this.balance += amount;
  }
}
```


The "**`this`**" parameter

- A way for an object to refer to itself inside a class.
- Used as if it was the object's name.

Local variables

- If you declare a variable inside of a method, it is a **local variable**.
- Stops existing as soon as the method finishes executing.
- Only accessible within the method.

Local variables continued

- The same thing applies to variables declared in compound statements.
- Example:

```
for (int i = 0; i <= 5; i++)
{
  int q = i*i;
}
System.out.println(q); // Invalid
```

Coordinate class

Recall the **Coordinate** class, used to get clicks from a 2D board:

```
Coordinate lastClicked = grid.getClick();
int r = lastClicked.getRow();
int c = lastClicked.getCol();
System.out.println("Last click at" +
           "(" + r + ", " + c + ")" );
```

A **Coordinate** has a row and column as its data.
Operations: getting the row and column.

A complete class

```
public class Coordinate {
   private int row, col;

   public Coordinate (int row, int col)
   {
      this.row = row;
      this.col = col;
   }

   public int getRow() { return row; }

   public int getCol() { return col; }
}
```

Summary

- Objects vs. classes
- Instance variables
- Methods
  - Invoking
  - Writing
  - Constructors
  - Local variables

ERROR: undefined
OFFENDING COMMAND:

STACK: