More on Objects and Classes

Roman Kontchakov
Birkbeck, University of London
Outline

- Object References
- Class Variables and Methods
- Packages
- Testing a Class
- Discovering Classes
  - Sections 8.7 – 8.11
- slides are available at
  www.dcs.bbk.ac.uk/~roman/sp1
Java Compilation

```
public static void main(String[] args) {
    ...
}

NB: statements must be inside methods!
```
Object-Oriented Programming

- Tasks are solved by collaborating **objects**
- Each object has its own set of encapsulated **data**, together with a set of **methods** that act upon the data (**public interface**)
- A **class** describes a set of objects with the same structure (i.e., data) and behaviour (i.e., methods)
- Encapsulation enables **changes in the implementation** without affecting users of the class
  - all instances variables should be private
  - *most* methods should be public
Example: Cash Register

class CashRegister {
    /* private data (instance variables) */
    private int itemCount;
    private double totalPrice;
    /* methods (public interface) */
    public void addItem(double price)
    {
        itemCount++; totalPrice += price;
    }
    public void clear()
    {
        itemCount = 0; totalPrice = 0;
    }
    public double getTotal() { return totalPrice; }
    public int getCount() { return itemCount; }
}

addItem(double) and clear() are mutators; getTotal() and getCount() are accessors
Using the CashRegister

// constructing objects
CashRegister reg1 = new CashRegister();
CashRegister reg2 = new CashRegister();

// invoking methods
reg1.addItem(1.95);
reg1.addItem(2.99);
System.out.println(reg1.getTotal() + " " + reg1.getCount());
reg2.addItem(7.67);
System.out.println(reg2.getTotal() + " " + reg2.getCount());
reg1.itemCount = 0; // COMPILATION TIME ERROR:
// private variable
Instance Variables

- Every instance of a class has its own set of instance variables

An object reference specifies the location of an object
Primitive Datatypes: Values are Copied

1 int num1 = 0;  
   num1 0

2 int num2 = num1;  
   num1 0  
   num2 0

3 num2++;  
   num1 0  
   num2 1

by executing num2++; only num2 is changed, num1 remains the same
Objects: References are Copied

1. `CashRegister reg1 = new CashRegister();`
   - `reg1`
   - `itemCount = 0`
   - `totalPrice = 0`

2. `CashRegister reg2 = reg1;`
   - `reg1`
   - `reg2`

3. `reg2.addItem(2.95);`
   - `reg1`
   - `reg2`
   - `itemCount = 1`
   - `totalPrice = 2.95`

`reg1` and `reg2` refer to the same object and so, all modifications by methods on `reg2` are reflected in `reg1`. 
The null Reference

The null reference refers to no object

```java
public static void printName(String firstName, String middleInitial, String lastName) {
    if (middleInitial == null)
        System.out.println(firstName + "   " + lastName);
    else
        System.out.println(firstName + "   " + middleInitial + "   " + lastName);
}

public static void test() {
    printName("Alice", "T", "A"); // output: Alice T A
    printName("Bob", null, "B"); // Bob B (1 space)
    printName("Ceri", ",", "C"); // Ceri   C (2 spaces)
}
```
The null Reference

It is an error to invoke a method on a null reference:

```java
1  CashRegister reg1 = null;
2  // RUN-TIME ERROR: null pointer exception
3  System.out.println(reg1.getTotal());
```

NB: without `null` the above fragment will not compile
local variables must be initialised before use
The this reference

In a method (or constructor) this refers to the object the method is called on

```java
public void addItem(double price) {
    this.itemCount++; // matter of taste
    this.totalPrice += price;
}
```

```java
public CashRegister() {
    this.clear(); // or simply clear();
}
```

**NB:** we shall see substantial uses of this later
A static variable belongs to the class, not to any object of the class.

```java
public class BankAccount {
    private double balance; // instance variable
    private int accountNumber; // instance variable
    // class variable
    private static int lastAssignedNumber = 1000;

    public BankAccount() {
        lastAssignedNumber++; // one for all instances
        accountNumber = lastAssignedNumber;
    }
}
```
1 `BankAccount a0 = new BankAccount();`  
    BankAccount a0  
    BankAccount.accountNumber = 1001  
    BankAccount.balance = 0.0  
    BankAccount.lastAssignedNumber = 1001

2 `BankAccount a1 = new BankAccount();`  
    BankAccount a1  
    BankAccount.accountNumber = 1002  
    BankAccount.balance = 0.0  
    BankAccount.lastAssignedNumber = 1002
public class BankAccount {
    public static final double OVERDRAFT_FEE = 29.95;
}

methods from any class can refer to this constant as
BankAccount.OVERDRAFT_FEE

NB: other examples include
    Math.PI: public static final double PI
    System.out: public static final PrintStream out
    javax.xml.XMLConstants.XMLNS_ATTRIBUTE:
        public static final String XMLNS_ATTRIBUTE
    ...
public class Financial {
    public static double percentOf(double percentage, double amount) {
        return (percentage / 100) * amount;
    }
}

class methods are not invoked on an object:
System.out.println(Financial.percentOf(50, 100));

NB: other examples include Math.abs, Math.sqrt, ...

NB: can one use this in a class method?
Packages

- A **package** is a set of related classes, e.g., `java.util`.
- The **import** directive lets you refer to a class from a package by its class name, without the package prefix.

1. `import java.util.Scanner; // before classes`
2. ...
3. `Scanner input = new Scanner(System.in);`
4. ...

Without this directive, one must use the full class name to import all classes of the package `java.util`, use

`import java.util.*`
Packages

- to put a class in a package, use
  ```
  package packagename;
  ```
  as the first statement in the source file

- use a domain name in reverse to construct
  an unambiguous package name: e.g.,
  ```
  uk.ac.bbk.dcs.sp1
  ```

- the path of a class file must match
  its package name: e.g.,
  ```
  uk.ac.bbk.dcs.sp1
  ```
  is looked up at
  ```
  uk/ac/bbk/dcs/sp1
  ```
  in the CLASSPATH directories
Overloading

Methods and constructors can have the same name (provided their signature (i.e., name and types of parameters) are different)

```java
public class CashRegister {
    // ...
    public CashRegister() {
        itemCount = 0;
        totalPrice = 0;
    }
    public CashRegister(CashRegister c) {
        this.itemCount = c.itemCount;
        this.totalPrice = c.totalPrice;
    }
}
```
Overloading (2)

1 // constructor 1: CashRegister()
2 CashRegister reg1 = new CashRegister();
3 reg1.addItem(2.95);

CashRegister
itemCount = 1
totalPrice = 2.95

4 // constructor 2: CashRegister(CashRegister)
5 CashRegister reg2 = new CashRegister(reg1);

CashRegister
itemCount = 1
totalPrice = 2.95

NB: type of arguments determines which constructor is called
Discovering Classes

- Find the concepts that you need to implement as classes. Often these will be nouns in the problem description.
- Find the responsibilities of the classes. Often these will be verbs in the problem description.
- Find relationships between the classes that you have discovered (aggregation, inheritance).

Class-Responsibility-Collaboration cards

subject of OO Analysis and Design
Take Home Messages

- encapsulation enables changes in implementation
  - all instances variables should be private
  - most methods should be public (public interface)

- every instance of a class has its own instance variables

- assignment statements copy
  - values for primitive datatypes
  - object references for object

- the null reference refers to no object
- static variables and methods belong to the class
- a package is a set of related classes