THE UNIVERSITY OF MELBOURNE

Programming and Software Development
COMP90041

Lecture 5

Immutable Objects

Dr. Thomas Christy
thomas.christy@unimelb.edu.au

Slides and materials were developed by Prof Peter Schachte
Copyright © The University of Melbourne

Objects

- Object oriented programming is centered around creating and using objects
- Each object is an instance of some class
- An object holds some data (state) and supports some operations
- *I.e.*, an object combines code and data
- An object represents some thing
  - Either a physical real world thing, like a student or a book or an airplane
  - Or a more abstract real world thing, like a university subject or a library loan or a flight
  - Or a still more abstract thing used for the purposes of a program, like a list or a string

Copyright © University of Melbourne 2018

Creating New Objects

- Java has special syntax for instantiating a class (creating a new object)
- Form: `new Class(expr1, expr2, ...)`
- This is an expression returning a fresh object
- Often used as:
  `Class var = new Class(expr1, expr2, ...)`
- We saw this for creating a `Scanner` object :
  `Scanner kbd = new Scanner(System.in);`
- The appropriate arguments are determined by the class's constructors (descussed below)
- You don't need to create `"constant strings"` — Java creates them for you

Copyright © University of Melbourne 2018

Using Objects

- Two ways to use an object:
  - Send a message to it
  - Pass it as an argument of a message to another object
- Form: `object.msg(expr1, expr2, ...))`
- Note the `expr` arguments can be objects
- We saw both uses of objects in our first program:

```
System.out.println("Hello, World!");
```

  - `System.out` is a `PrintStream` object connected to the console
  - `"Hello, World!"` is a constant `String` object
- The allowed messages (`msg`) are determined by the class of the object (`object`) receiving the message

Copyright © University of Melbourne 2018

null

- `null` is a special value that means "there is no object here"
- Java lets you use it wherever an object can be used
- But is not an object, and causes an error if you use it as one
  - You cannot send a message to it, or access its instance variables (it has none)
- Be sure a variable is not `null` before sending it a message, *e.g.*:

```
String s = ...
if (s != null) {
    System.out.println("the string is " + s);
}
```

Copyright © University of Melbourne 2018

Modelling

- Software modelling: determining an appropriate set of classes and methods to solve a problem
- Best practice: design classes around the "entities" of your program (concrete or abstract)
  - What characteristics are different for different individuals?
  - What can they do or have done to them?
  - Ignore what is irrelevant to your application
- Best practice: Keep It Simple
- These sometimes conflict: use judgement

Copyright © University of Melbourne 2018

Modelling Example: Reuniting Loved Ones

- Problem: in a natural disaster, people may be separated from their loved ones
- Software solution, each person registers where they are, and when, and who they're looking for
- Need to model person, location, and date/time
- Person has a name and location
- People may come and go, so need to know when last seen at that location
- Represent name as a single string? Separate given and family name? Include middle name? Title?
- Similarly for location and date
- Go with simplest design that gives the information you need

Copyright © University of Melbourne 2018

Modelling Operations

- Consider what the application needs to do with an object
- In reuniting example, person enters their name and location in system, and the name of their missing loved one
- Loved one in other location does the same
- So program needs to compare names of people to match them up
- Need a fuzzy match, eg, "Bill" = "William"
- Need to report person's location and time when a match is found

Copyright © University of Melbourne 2018

Classes

- Implement a design by defining classes
- Classes contain:
  - Instance variables, which hold the data of an object
  - (Instance) methods, which define the operations (code) of an object
- A class is a type, which can be used to declare variables that hold instances of that type
- Each object has its own value for each instance variable
- All objects of a class have the same methods
- Each method of a class specifies an operation on that class and how it should behave

Copyright © University of Melbourne 2018

Declaring Instance Variables

- Declare instance variable inside class, but outside any method
- Form: `private` *`type name`* `;`
- Like class variable declaration without `static`
- Declares instance variable *`name`* holding value of type *`type`*
- *E.g.*:

```
public class Person {
    private String familyName;
    private String givenName;
    ⋮
}
```

Copyright © University of Melbourne 2018

Make Instance Variables Private

- It's possible to declare instance variables `public` instead of `private`
- But **don't**
- Making an instance variable `public` would allow any method of any class to modify it
- The class would lose control of its own data!
- Instead use methods to access instance variables. . .

Copyright © University of Melbourne 2018

Assigning and Using Instance Variables

- Inside a class's methods, its instance variables can be assigned and used like other variables: by name
- Local variables live in a method; class variables live in a class; instance variables live in an object
  - local variables only live while method is executing
  - instance variables live as long as object is around
  - class variables live as long as the program is running
  - local variables must be initialised every time method executes
  - instance variables must be initialised when declared or when object is created
  - class variables must be initialised where declared, which is executed when program starts
- If local and instance variables with same name, local variable "wins"

Copyright © University of Melbourne 2018

Declaring (Instance) Methods

- Declare instance (non-static) methods inside class, but outside any method
- Form:

```
vis type name(type1 name1,...) {
    ⋮
}
```

- Like class method declaration without `static`
- *vis* can be `public` (use from any class) or `private` (use only from same class)
- Declares instance method *name* returning value of *type*, taking arguments of type *type1*,...

Copyright © University of Melbourne 2018

Example Method

```
public class Person {
    private String familyName;
    private String givenName;
    ⋮
    public String getFullName() {
        return familyName + ", " + givenName;
    }
    ⋮
}
```

- Each `Person` object has its own `familyName` and `givenName`
- given a `Person` object `p`, `p.getFullName()` returns that person's full name as a string

Copyright © University of Melbourne 2018

Quick Poll

qp.unimelb.edu.au/tchristy

Accessible on any electronic device!

Quick Poll

What does this print?

```
public class QP0501 {
    private int x=1, y=2;
    public int m(int y) { return x + y; }
    public static void main(String[] args) {
        QP0501 ob = new QP0501();
        System.out.println(ob.m(3));
    }
}
```

A. 3
B. 4
C. 5
D. Won't compile: `m` cannot access `x`
E. Won't compile: two different `y`s

Copyright © University of Melbourne 2018

What does this print?

```
public class QP0501 {
    private int x=1, y=2;
    public int m(int y) { return x + y; }
    public static void main(String[] args) {
        QP0501 ob = new QP0501();
        System.out.println(ob.m(3));
    }
}
```

A. 3
B. 4
C. 5
D. Won't compile: `m` cannot access `x`
E. Won't compile: two different `y`s

Copyright © University of Melbourne 2018

Accessors (Getters)

- If you declare instance variables `private`, how can you access them?
- An accessor (getter) is a very simple method written for this purpose
- Java convention: name accessor after instance variable, with `get` on the front

```
public String getFamilyName() {
    return familyName;
}
```

- Only write accessors for instance variables you really need other classes to be able to access

Copyright © University of Melbourne 2018

Quick break

Constructors

- When creating an object, its instance variables need to be initialised to appropriate values
- Constructors are special methods responsible for this
- Form:

```
public ClassName(type1 var1,...) {
    ⋮
}
```

- Constructors always have same name as class
- And never specify a return type, not even `void`

Copyright © University of Melbourne 2018

Using a Constructor

- The constructor body can use the parameters to initialise the instance variables

```
public Person(String f, String g) {
    familyName = f;
    givenName = g;
}
```

- Use a constructor by passing the right number and types of arguments

```
Person king = new Person("Snow", "Jon");
```

- Cannot use `new` without a matching constructor
- Java generates a do-nothing constructor with no arguments for classes with no constructors

Copyright © University of Melbourne 2018

Initialisation - Default Values

- Constructor
  - `ClassName()`
- toString
  - `toString()`
- Variables
  - Some flavour of `zero`
    - `0, 0.0, "", false, etc.`
- Class
  - `null`

Copyright © University of Melbourne 2018

Assigning and Using Instance Variables

- Instance variables can be assigned and used just like local variables
- If an instance variable and a local variable have the same name, the local variable "wins"
- Can use an alternative form: `object.name`
- Refers to the `name` instance variable of `object`
- Works if `object` is an instance of the class this code appears in
- ...or if `name` is a public instance variable (which it shouldn't be)

Copyright © University of Melbourne 2018

this

- Inside a method, special keyword `this` always holds the object the current message was sent to
- An implicit parameter to instance methods
- *E.g.*, if message was `foo.bar(baz)`, while executing method `bar`, `this` is `foo`
- Inside constructor, `this` is the object being created

```
public Person(String givenName,
              String familyName) {
    this.givenName = givenName;
    this.familyName = familyName;
}
```

Copyright © University of Melbourne 2018

Null Pointer Exception

- If you forget to check that an object is not `null` before sending it a message or accessing its instance variables, you will get a null pointer exception
- This aborts (crashes) your program

```
Exception in thread "main" java.lang.NullPointerExcepti
on
        at Tst.main(Tst.java:4)
```

- But methods don't need to check that `this` is not `null`: if it were, the method would not run
- If you checked a variable, don't need to check again unless it could have been changed

Copyright © University of Melbourne 2018

Immutable Objects

- Immutable objects cannot be changed once they are created
- An Immutable class is one all of whose instances are immutable
- To be immutable, a class must not have any public methods that modify the object (just constructor)
- In Java, the `String` class is immutable
- **Primitive types** are also immutable
- Bad Things can happen when you modify objects (as we will see)
- Best practice: make classes immutable if possible

Copyright © University of Melbourne 2018

Finality

- Java can help you make classes immutable
- Add `final` keyword to instance variables before type
- Then Java will only let you set it once

```
public class ImmutablePerson {
    private final String familyName;
    private final String givenName;

    public ImmutablePerson(String familyName,
            String givenName) {
        this.familyName = familyName;
        this.givenName = givenName;
    }

   ⋮
```

Copyright © University of Melbourne 2018

Finality Protects You From Yourself

Adding this method:

```
    public void bogus(String s) {
        this.familyName = s;
    }
```

gives this compiler error:

```
ImmutablePerson.java:12: error: cannot assign
 a value to final variable familyName
        this.familyName = s;
            ^
1 error
```

Copyright © University of Melbourne 2018

Class vs. Instance Messages

- An instance message is sent an individual object
- A class message is sent to the whole class
  - Not to any individual object
- So a class method does not have (cannot use) `this`
- Class method cannot access instance variables!
- Class methods cannot use instance methods (without specifying *`object.`* in front)
- but instance methods can use class methods

Copyright © University of Melbourne 2018

The `toString` method

- *E.g.*, if `p` is an `ImmutablePerson`, what should `"Welcome, " + p` produce?
- What should `System.out.print(p)` print?
- You can determine how Java converts instances of your class to a String
- Define a public method `String toString()`

```
public String toString() {
    return givenName + " " + familyName;
}
```

Copyright © University of Melbourne 2018

Summary

- Use `new Class(...)` to create object
- Use `object.methodName(...)` to send message to `object`
- Can use `null` in place of an object, but don't send message to it!
- Declare instance variables and methods like class ones, without `static` keyword
- Always make instance variables private
- Write constructor to initialise instance variables
- Make classes immutable when you can

Copyright © University of Melbourne 2018

THE UNIVERSITY OF
MELBOURNE

© Copyright The University of Melbourne 2018