Menti Q&A QR

Object Orientated Programming (Recap)

Blair Archibald

Setting the Speed

- Quick show of hands:

Setting the Speed

- Quick show of hands:
  - Who has done Object Orientated Programming (OOP) before?

Setting the Speed

- Quick show of hands:
  - Who has done Object Orientated Programming (OOP) before?
  - Who knows what inheritance is? (could explain it to someone else)

Setting the Speed

- Quick show of hands:
  - Who has done Object Orientated Programming (OOP) before?
  - Who knows what inheritance is? (could explain it to someone else)
  - Who knows what polymorphism is? (could explain it to someone else)

Object Orientated Programming

OOP is about keeping **data** and the **methods** that act on that data **together**

- Classes are *specifications*
  - What data is specified
  - What methods are specified
- Objects are *instances* of a class
  - Exist in memory
  - Independent of other objects of the same class

Object Orientated Programming

- Objects can be anything, but usually represent domain entities:
  - Ledger
  - BankAccount
  - Scene
  - List
  - Integer
  - HTMLFormatter

Defining Classes and Objects in Java

```java
public class Point {
  private float x;
  private float y;

  public Point(float x, float y) {
     this.x = x;
     this.y = y;
  }

  public String toString() {
    return "(" + x + "," + y + ")"
  }
}
```

```java
Point p1 = new Point(2.0, 4.0);
Point p2 = new Point(2.2, 6.7);
System.out.println(p1.toString());
```

Communication Between Objects

- Objects *interact* by calling functions[1] on other objects
  - Possibly passing some data
- Programs are completely specified by interacting objects
  - Especially true in Java: *everything is an object*
  - One class has a special "main" (static) function

[1] Traditionally objects "passed messages" to each other

A Mental Model for Objects

(Mental Model: Real implementations are *slightly* smarter!)

- You can think of objects as tables
- Map identifiers to:
  - Location of data
  - Location of a function
- Classes are the *schema* for these tables
  - E.g. what elements it must have
- Can "lock" some elements
  - "public"/"private"

Identifier	Location
"x"	Memory for x
"y"	Memory for y
"toString"	Function for toString

Inheritance: Extending Classes

- Add new functionality to existing classes
  - Avoid duplicating code
  - Store additional data
  - Override behaviour
- Extended class shares:
  - All public/protected data and functions
  - **not** private data/functions
  - The *type* of the parent
- Java does not support multiple inheritance[2]

[2] Unless your class meets the special conditions to be an *interface* (later in the course)

Inheritance Example: Labelled Points

```java
public class LabelledPoint extends Point {
  private String label;

  public LabelledPoint(float x, float y, String lbl) {
    // Call parent constructor to initialise x and y
    // Can't do that manually here because they were
    // private!
    super(x,y);
    label = lbl;
  }

  // Override to new functionality
  public String toString() {
    return "(" + x + "," + y + ") lbl: " + lbl
  }
}
```

```java
Point p1 = new Point(2.0, 4.0);

LabelledPoint p2 =
  new LabelledPoint(2.2, 6.7, "Test");

System.out.println(p2.toString());
```

Inheritance Typical Use

- Base class:
  - Defines an interface for set of sub-classes
  - Specific features added to sub-classes
  - System only interacts through base class interface

Mental Model for Inheritance

Inheritance adds new rows! (possibly changing functions if needed)

Identifier	Location
“x”	Memory for x
“y”	Memory for y
“toString”	Function for toString

Identifier	Location
“x”	Memory for x
“y”	Memory for y
“toString”	**New** toString function
“label”	Memory for label

Polymorphism 

- Sub-classes share the type of the parent
  - you can't *remove* features only *extend* (tables only get bigger)
- Means they can be used *transparently* wherever a parent could be

```
public void myFunc(Point p) {
  System.out.println(p.toString());
}

Point p1 = new LabelledPoint(2.2, 6.7, "Test");  这是一种新的实例化方法.
myFunc(p1) // Does the "right" thing
```

- `Point p` may have many different forms underneath
  - Polymorphism = Many forms

Polymorphism: Behaviour Depending on Types

Polymorphism means an objects concrete behaviour depends on it's type

- Previous example
  - The correct `toString()` was chosen

Polymorphism: Hiding Behaviour

Polymorphism is equally useful because of what you **can't** do

Polymorphism: Hiding Behaviour

Polymorphism is equally useful because of what you **can't** do

A caller is *not* allowed to know the exact type of an object
Only that it has *at least* the functionality of the given (general) type

I.e. You cannot assume any specific behaviour unless you use a specific type

Polymorphism Example

```
public class LabelledPoint extends Point {
   public String getLabel() { return label; }
   ...
```

```
LabelledPoint p = new LabelledPoint(2.0,2.0,"test");
// Fine because p has (at least) the functionality of LabelledPoint
String s = p.getLabel()

Point p2 = new LabelledPoint(2.0,2.0,"test");
// Not allowed since Points don't have a getLabel
String s = p2.getLabel();
```

Polymorphism Example

- Overriden behaviours are still maintained

```
LabelledPoint p = new LabelledPoint(2.0,2.0,"test");
String s = p.toString()
// (2.0,2.0) lbl: test
```

We separate what the callers are allowed to know about from actual behaviour

Mental Model for Polymorphism

Polymorphism works because: two objects with compatible tables (types) can point to different functions

Identifier	Location
"x"	Memory for x
"y"	Memory for y
"toString"	Function for toString $\rightarrow$ F1

Identifier	Location
"x"	Memory for x
"y"	Memory for y
"toString"	toString function $\rightarrow$ F2
"label"	Memory for label

Program only knows it should: "call the function the table gives me for toString()"

Four Pillars of OOP

- **Abstraction**
  - Implementation details hidden behind interfaces (function calls)
  - We might not know the exact function (polymorphism)
- **Encapsulation**
  - Objects control their own state, e.g. they have data inside
    - Ideally hidden/well controlled access to it only
- **Inheritance**
  - Objects are open to extension
- **Polymorphism**
  - *Exact* forms of objects can be hidden

Summary

We will see a lot more OOP later in the course

- Core idea:
  - Data + Methods acting on data (behaviours)
  - Ways to **extend** behaviour
  - Ways to **transparently** change behaviour

Summary

We will see a lot more OOP later in the course

- Core idea:
  - Data + Methods acting on data (behaviours)
  - Ways to **extend** behaviour
  - Ways to **transparently** change behaviour

The Challenge: how we combine these objects to solve problems! (**This Semester!**)