Object Oriented Programming

Programación II
Facultad de Ingeniería
Universidad Austral

Object Oriented Programming tries to help with scale

- **Writing programs gets harder as**
  - the program gets bigger
  - the team gets bigger
- **Today's programs are massive**
- **Object Oriented Programming is a style of programming**
- **The intention is to make it easier for your programming to scale**

Books

- **OOP Concepts:**
  - Java: How to Program by Deitel & Deitel
  - Thinking in Java by Eckels
  - Java in a Nutshell (O' Reilly) if you already know another OOP language
  - Java specification book: http://java.sun.com/docs/books/jls/
  - Design Patterns by Gamma et al.
- **Non-encyclopedic (funy) books**
  - Effective Java by Joshua Bloch
  - Java Puzzlers by Joshua Bloch

Types of Languages

- **Declarative - specify what to do, not how to do it. i.e.**
  - E.g. HTML describes what should appear on a web page, and not how it should be drawn to the screen.
  - E.g. SQL statements such as “select * from table” tell a program to get information from a database, but not how to do so.
- **Imperative – specify both what and how**
  - E.g. “triple x“ might be a declarative instruction that you want the variable x tripled in value. Imperatively we would have “x=x*3” or “x=x+x+x”

Functional Language

- **Functional languages are a subset of declarative languages**
  - Haskel is a functional language
  - It may appear that you tell it how to do everything, but you should think of it as providing an explicit example of what should happen
- **The compiler may optimise i.e. replace your implementation with something entirely different but 100% equivalent.**

```
let rec factorial n =
    match n with
        | 0 -> 1
        | 1 -> 1
        | n -> n * (factorial (n – 1));
```

Function Side Effects

- **Functions in imperative languages can use or alter larger system state → *procedures***

Java:

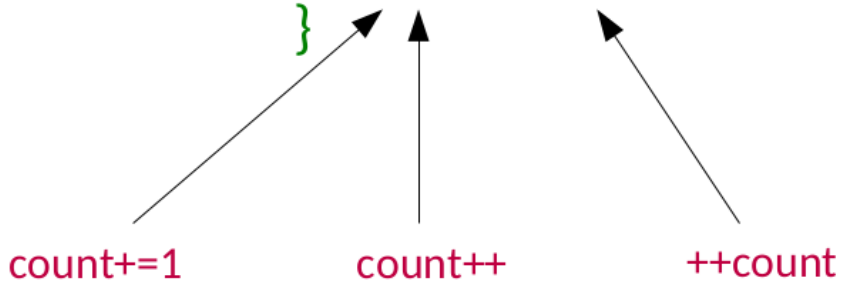
```
int y = 7;
int m(int x) {
        y=y+1;
        return x*y;
}
```

void Procedures

- **A void procedure returns nothing:**

```
int count=0;

void addToCount() {
  count=count+1;
}
```



Control Flow: Looping

for(*initialisation; termination; increment* **)**

```
for (int i=0; i<8; i++) ...

int j=0;  for(; j<8; j++) ...

for(int k=7;k>=0; j--) ...
```

Demo: printing the numbers from 1 to 10

while(*boolean_expression* **)**

```
int i=0;  while (i<8) { i++; ...}

int j=7; while (j>=0) { j--; ...}
```

Control Flow: Branching I

- **Branching statements interrupt the current control flow**

return

- Used to return from a function at any point

```
boolean linearSearch(int[] xs, int v) {
    for (int i=0;i<xs.length; i++) {
        if (xs[i]==v) return true;
    }
    return false;
}
```

Control Flow: Branching II

- **Branching statements interrupt the current control flow**

break

- Used to jump out of a loop

```
boolean linearSearch(int[] xs, int v) {
    boolean found=false;
    for (int i=0;i<xs.length; i++) {
        if (xs[i]==v) {
            found=true;
            break; // stop looping
        }
    }
    return found;
}
```

Control Flow: Branching III

- **Branching statements interrupt the current control flow**

continue

- Used to skip the current iteration in a loop

```
void printPositives(int[] xs) {

     for (int i=0;i<xs.length; i++) {
               if (xs[i]<0) continue;
               System.out.println(xs[i]);
     }
}
```

Immutable to Mutable Data

- **Java is a language of statements and expressions**

Java

```
int x=5;
x=7;

int x=9;
for(int i=0;i<10;i++) {
  System.out.println(i);
}
```

Evaluates to the value 7 with type int

Does not evaluate to a value and has no type

Demo: returning vs printing

Types and Variables

- **Java and C++ have limited forms of type inference**

```
var x = 512;
int y = 200;
int z = x+y;
```

- **The high-level language has a series of primitive (built-in) types that we use to signify what's in the memory**
  - The compiler then knows what to do with them
  - E.g. An "int" is a primitive type in C, C++, Java and many languages. In Java it is a 32-bit signed integer.
- **A variable is a name used in the code to refer to a specific instance of a type**
  - x,y,z are variables above
  - They are all of type int

E.g. Primitive Types in Java

- **“Primitive” types are the built in ones.**
  - They are building blocks for more complicated types that we will be looking at soon.
- **boolean – 1 bit (true, false)**
- **char – 16 bits**
- **byte – 8 bits as a signed integer (-128 to 127)**
- **short – 16 bits as a signed integer**
- **int – 32 bits as a signed integer**
- **long – 64 bits as a signed integer**
- **float – 32 bits as a float point number**
- **double – 64 bits as a float point number**

Widening vs Narrowing

Overloading Functions

- **Same function name**
- **Different arguments**
- **Possibly different return type**

```
int myfun(int a, int b) {...}
float myfun(float a, float b) {...}
double myfun(double a, double b) {...}
```

- **But <u>not</u> just a different return type**

```
int myfun(int a, int b) {...}
float myfun(int a, int b) {...}
```

X

Function Prototypes

- **Functions are made up of a prototype and a body**
  - Prototype specifies the function name, arguments and possibly return type
  - Body is the actual function code

```
int myfun(int a, int b) {...}
```

Custom Types

```
public class Vector3D {
        float x;
        float y;
        float z;
}
```

State and Behaviour

```
public class Vector3D {
      float x;
      float y;
      float z;
      void add(float vx, float vy, float vz) {
            x=x+vx;
            y=y+vy;
            z=z+vz;
      }
}
```

STATE

BEHAVIOUR

Terminology

State	Behaviour
Fields	Functions
Instance Variables	Methods
Properties	Procedures
Variables	
Members	

Classes, Instances and Objects

- **Classes can be seen as templates for representing various concepts**
- **We create instances of classes in a similar way. e.g.**

```
MyCoolClass m = new MyCoolClass();
MyCoolClass n = new MyCoolClass();
```

makes two instances of class MyCoolClass.

- **An instance of a class is called an object**

Defining a Class

```
public class Vector3D {
    float x;
    float y;
    float z;
    void add(Hoat vx, Hoat vy, Hoat vz) {
        x=x+vx;
        y=y+vy;
        z=z+vz;
    }
}
```


Questions?