

Java Basics

Object Orientated Programming in Java

Benjamin Kenwright

Outline

■ Essential Java Concepts

- ▷ Syntax, Grammar, Formatting, ...
- ▷ Introduce Object-Orientated Concepts
 - Encapsulation, Abstract Data, OO Languages,...

■ Today's Practical

■ Review/Discussion

Last Week

- Compile Java Programs

 - ▷ Javac.exe/Java.exe

- Setup IDE

- Basic Programs

 - ▷ Hello World

- Simple Debugging

 - ▷ e.g., Program entry point, hello worlds, print out (println..)

- Read Chapters 1 & 2

Question

☐ Java is case sensitive?

☐ A. True

☐ B. False

Answer

A. True

Question

☐ What will be output of x in following code?
"class Test{ public static void main(String[] args) { int x = 1; if (x == 1) { x = x + 1} } }"

☐ A. 0

☐ B. 1

☐ C. 2

☐ D. 3

☐ E. Compile Error

Answer

■ E. Compile Error

Missing semi-colon (;)

Question

■ What will be output of x in following code?
"class Test{ public static void main(String[] args) { int x = 1; if (x == 1) { x = x + 1;} } }"

■ A. 0

■ B. 1

■ C. 2

■ D. 3

■ E. Compile Error

Answer

■ C. 2

Today

■ Exercises from Chapters 2, 4, 5 and 6

- ▷ Data types (boolean, int, string, ..)
- ▷ Loops (while, for, ...)
- ▷ Conditional Logic (if, else, switch, ..)
- ▷ Math libraries
- ▷ Arrays
- ▷ Methods (calling and passing parameters)

Pure Object-Oriented Language

- ***Everything is an object***
- A program is a set of objects telling each other what to do by sending messages
- Each object has its own memory (made up by other objects)
- Every ***object has a type***
- All objects of a specific type can receive the same messages

Java breaks some of these rules in the name of efficiency

Object Concept

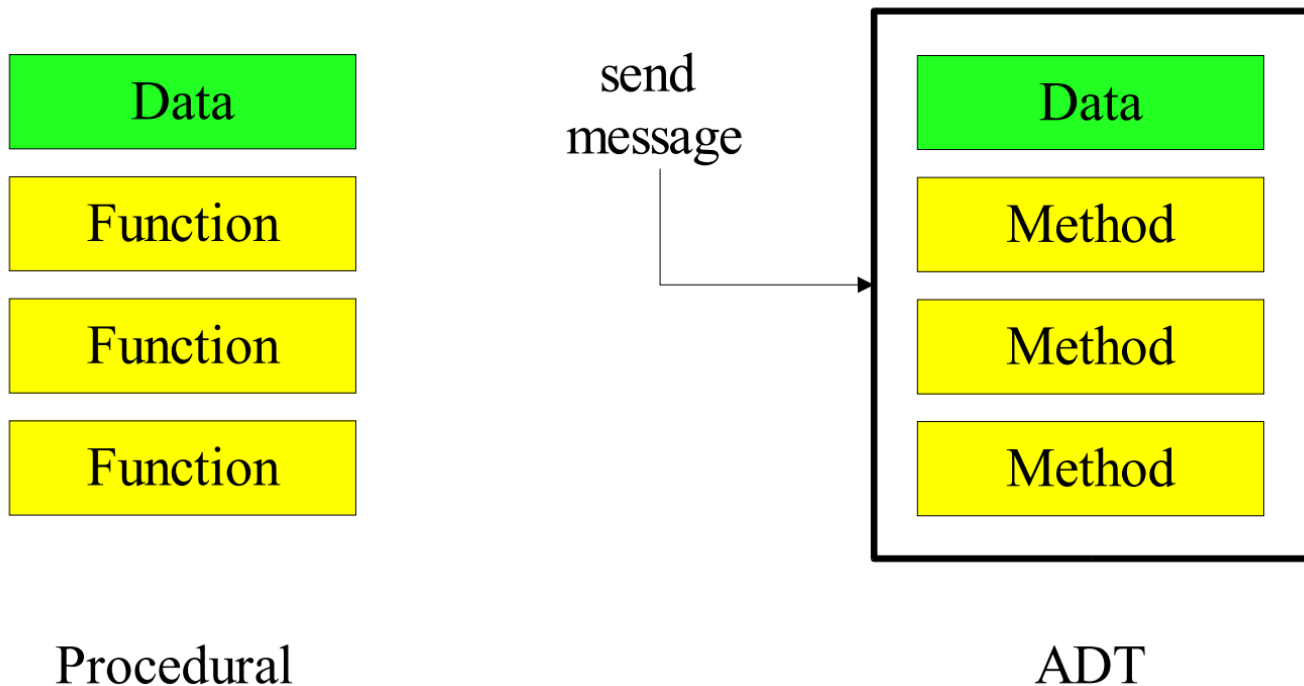
- An object is an *encapsulation* of data
- An object has
 - ▷ identity (a unique reference),
 - ▷ state, also called characteristics
 - ▷ behavior
- An object is an instance of an *abstract data type*
- An abstract data type is implemented via a *class*

Abstract Data Type (ADT)

- An ADT is a *collection* of objects (or values) and a corresponding set of methods
- An ADT *encapsulates the data* representation and makes data access possible at a higher level of abstraction
- Example 1: A set of *vehicles* with operations for starting, stopping, driving, get km/litre, etc
- Example 2: A *time-interval*, start time, end time, duration, overlapping intervals, etc

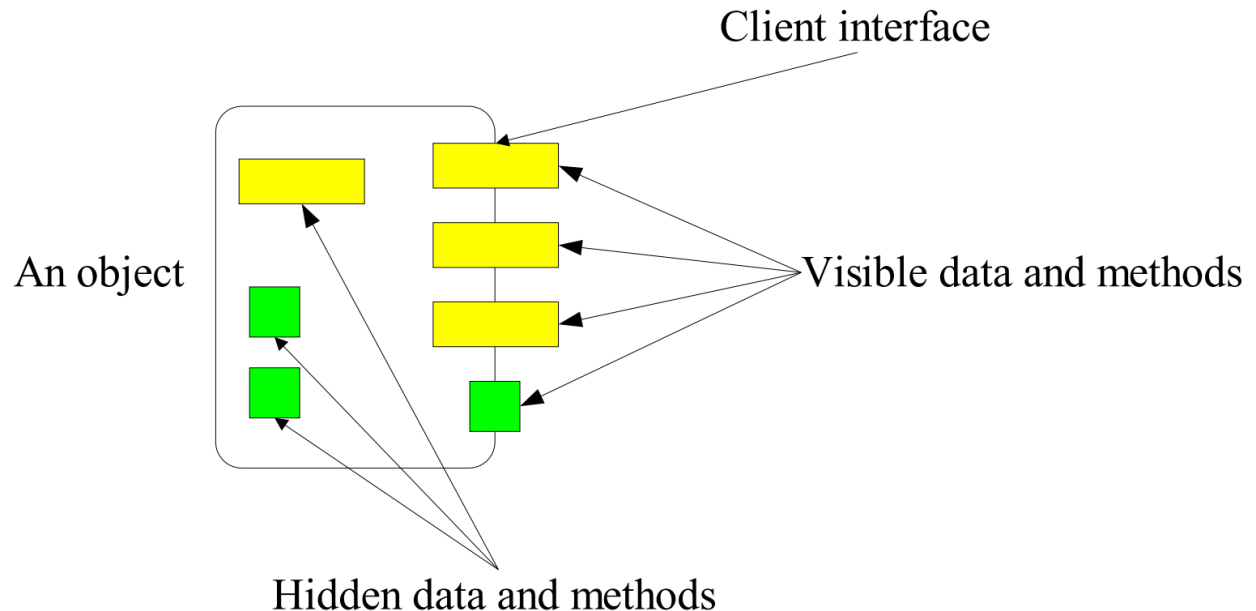
Encapsulation and Information Hiding

- Data can be encapsulated such that it is *invisible to the "outside world"*
- Data can only be *accessed via methods*



Encapsulation and Information Hiding

- What the "*outside world*" *cannot* see it cannot depend on!
- "*Wall*" between the object and the "outside world"
- The *hidden data and methods can be changed without affecting the "outside world"*



Class vs. Object

Class

- A description of the *common properties* of a set of objects
- A concept
- A *class* is a part of a program
- Example 1: Person
- Example 2: Album

Object

- A representation of the *properties* of a single *instance*
- A phenomenon
- An *object* is part of data and a program execution
- Example 1: Bill Clinton, Bono, Viggo Jensen
- Example 2: A Hard Day's Night, Joshua Tree

Type and Interface

- An object has type and an interface

Account	Type
balance () withdraw () deposit ()	Interface

- To get an object: *Account a = new Account()*
- To send a message: *a.withdraw()*

Instantiating Classes

- An instantiation is a mechanism where *objects* are *created from a class*
- Always involves storage *allocation* for the *object*
- A mechanism where objects are given an *initial state*

Static Instantiating

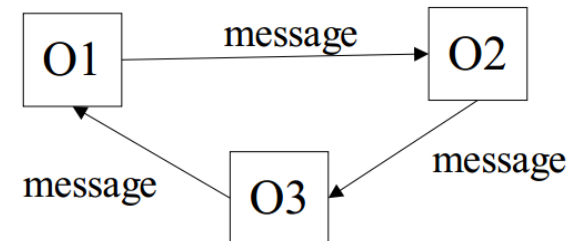
- In the declaration part of a program
- A static instance is *implicitly created*

Dynamic Instantiating

- In the method part of a program
- A dynamic instance is *created explicitly* with a special command

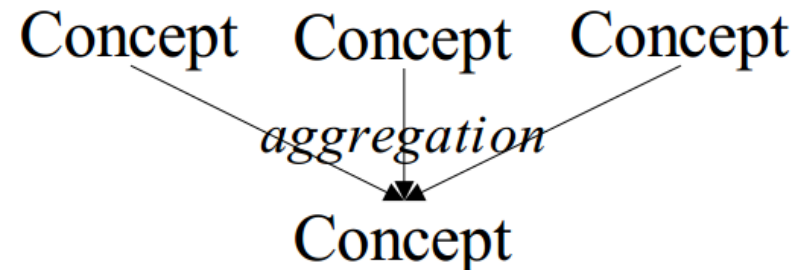
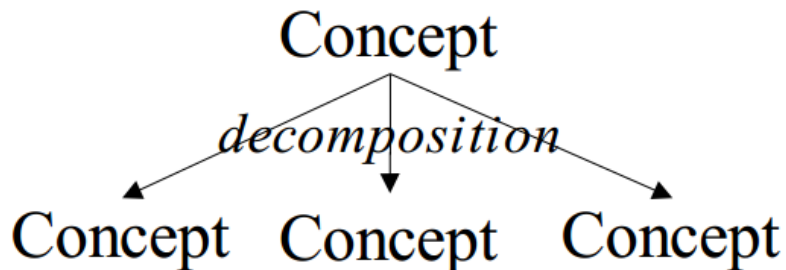
Interaction between Objects

- *Interaction between objects* happens by *messages* being send
 - ▷ A message activates a method on the calling object
- An object O1 interacts with another object O2 by calling a method on O2
 - ▷ “O1 sends O2 a message”
- The call of a method *corresponds to a procedure call* in a non object-oriented language such as C or Pascal



Aggregation and Decomposition

- A *decomposition* splits a single concept into a number of (sub-)concepts
- An *aggregation* consists of a number of (sub-)concepts which collectively is considered a new concept



Aggregation and Decomposition, Example

- Idea: make *new objects* by *combining existing objects*
- *Reusing* the implementation

Car
start() drive()
Engine Gearbox Doors [4]

new class

Engine
start() stop()

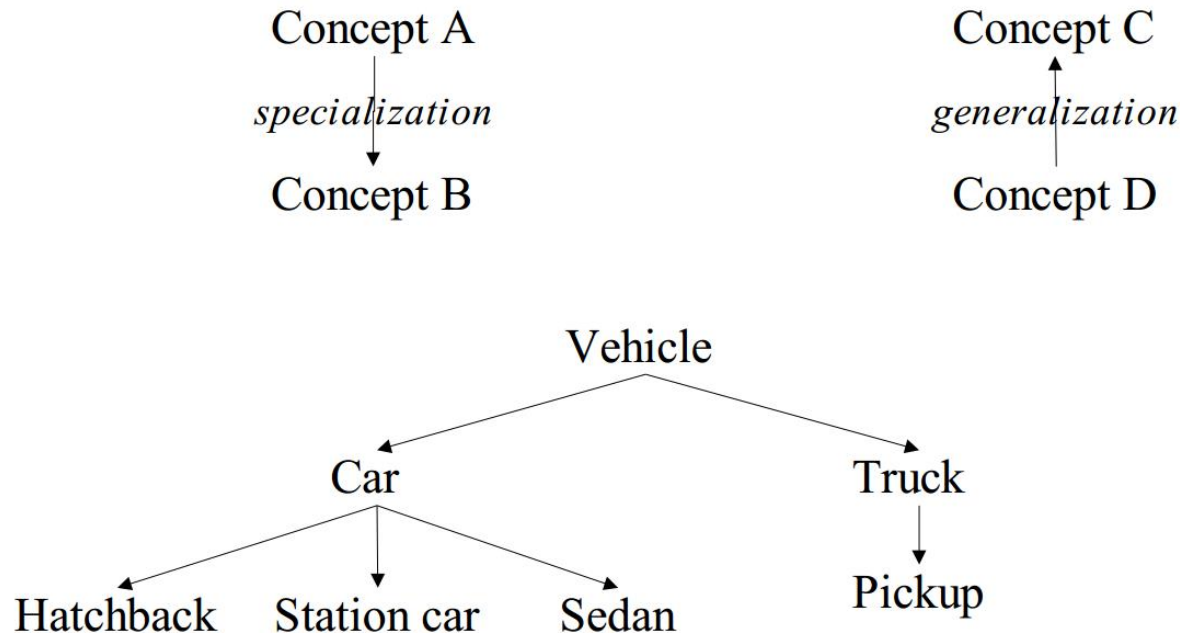
Gearbox
up() down()

Door
open() close()

existing classes

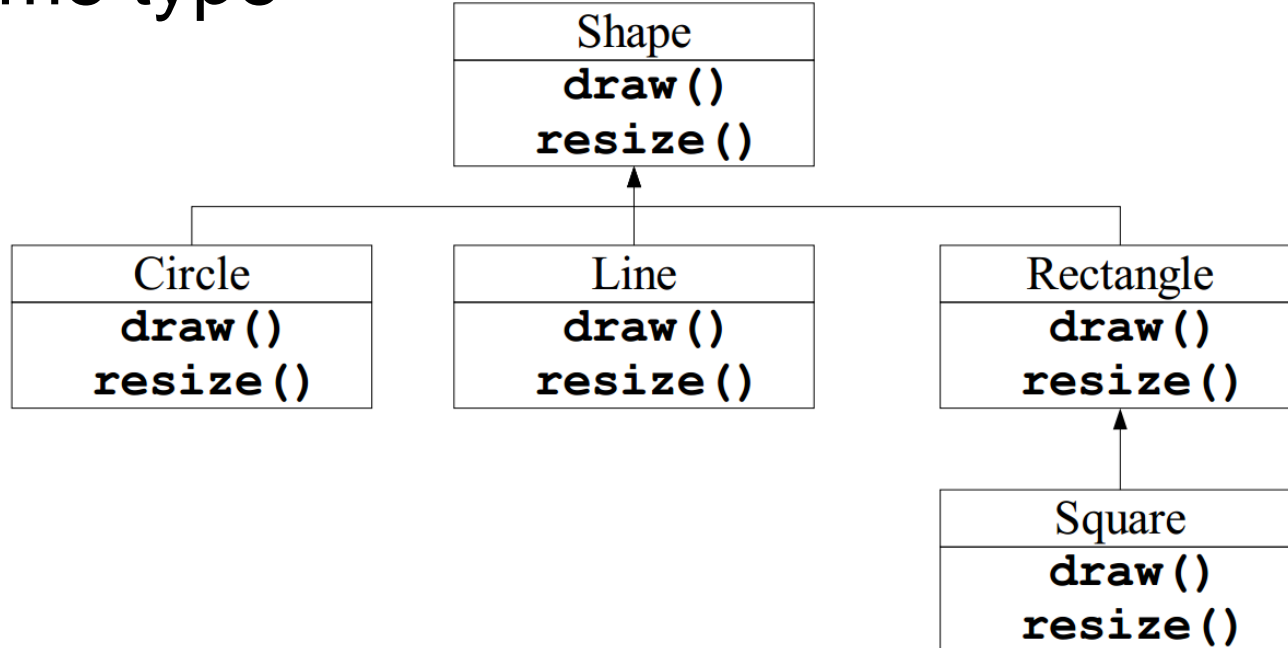
Generalization and Specialization

- *Generalization* creates a concept with a *broader* scope
- *Specialization* creates a concept with a *narrower* scope
- Reusing the interface



Generalization and Specialization, Example

- **Inheritance**: get the interface from the general class
- Objects *related by inheritance* are all of the same type



Code Example

- *Polymorphism*: One piece of code works with *all shape* objects
- *Dynamic binding*: How polymorphism is implemented

```
void doSomething(Shape s){  
    s.draw(); // "magically" calls on specific class  
    s.resize();  
}  
  
Circle c = new Circle();  
Line l = new Line();  
Rectangle r = new Rectangle();  
  
doSomething(c);           // dynamic binding  
doSomething(l);  
doSomething(r);
```


Structuring by Program or Data?

- What are the actions of the program vs. which data does the program act on
 - ▷ *Top-down*: Stepwise program refinement
 - ▷ *Bottom-up*: Focus on the stable data parts then add methods
- *Object-oriented programming is bottom-up*. Programs are structured with outset in the data
- C and Pascal programs are typically implemented in a more *top-down* fashion

Review Java Program Structure

```
// comment on the class
```

```
public class MyProg {  
    String s = "Viggo";
```

variable



```
/**
```

```
 * The main method (comment on method)
```

```
 */
```

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

```
    // just write some stuff
```

```
    System.out.println ("Hello World"); }
```

method header



method body



```
}
```

Java Class Example Car

```
/** A simple class modeling a car. */
public class Car {
    // instance variables
    private String make;  private String model;
    private double price;

    // String representation of the car
    public Car(String m, String mo, double p) {
        make = m; model = mo; price = p;
    }

    // String representation of the car
    public String toString() {
        return "make: " + make + " model: "
            + model + " price: " + price;
    }
}
```

Question

☐ Is Java a `top-down` or `bottom-up` programming language?

☐ A. `top-down`

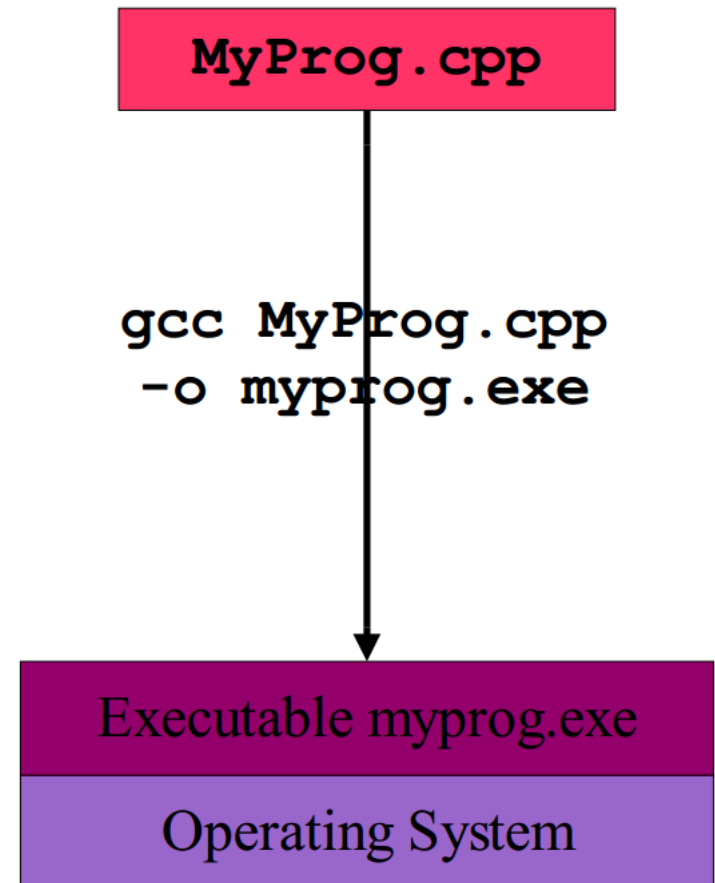
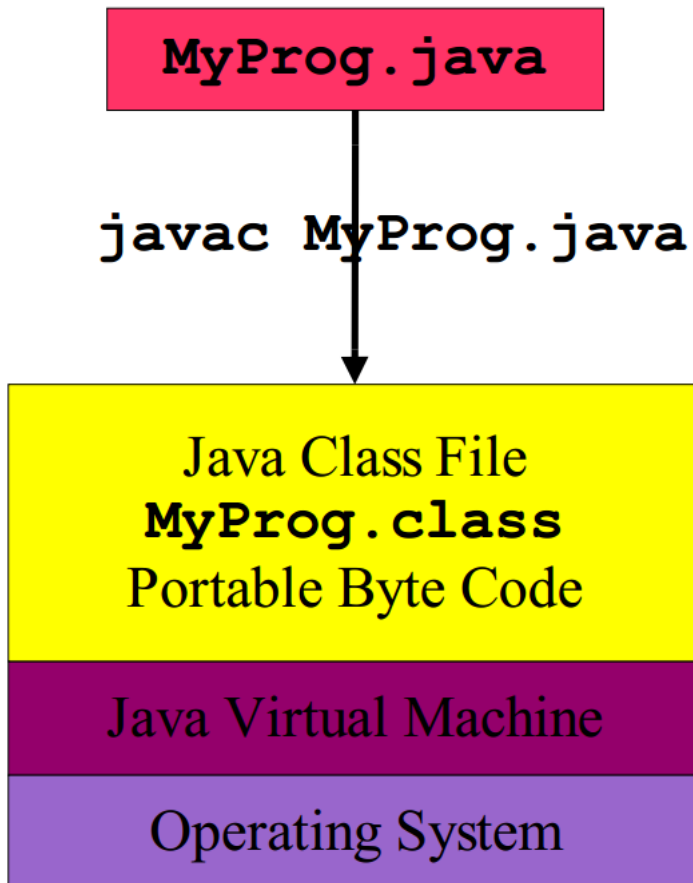
☐ B. `bottom-up`

Answer

■ B. 'bottom-up'

Object-oriented programming is bottom-up. Programs are structure with outset in the data

Byte Code vs. Executable



Difference from C/C++

- *Everything* resides in a *class*
 - ▷ variables and methods
- *Garbage* collection
- Error and exception handling
- *No global* variables or methods
- *No* local *static* variables
- No separation of declaration and implementation (*no header files*).
- *No* explicit *pointer* operations (uses references)
- *No pre-processor* (but something similar)
- Has fewer "dark corners"
- Has a much larger standard library

Question

■ What displays from the following statements?
String word = "abcde"; for (int i = 0; i <4; i+=2)
System.out.print(word.charAt(i));

- A. ab
- B. ac
- C. ace
- D. bd

// access characters in a String using charAt(i)
similar to word[i] in C language

Answer

☒ B. ac

Review Concepts

- *Classes* are "*recipes*" for creating objects
- All objects are *instances of classes*
- An ADT is implemented in a class
- *Aggregation and decomposition*
 - ▷ "has-a" relationship
- *Generalization and specialization*
 - ▷ "is-a" or "is-like-a" relationship
- *Encapsulation*
 - ▷ Key feature of object-oriented programming
 - ▷ Separation of interface from implementation
 - ▷ It is not possible to access the private parts of an object

This Week

- Read Chapters 3, 4, 5, 6
- Review Slides
- Complete Java Chapter Exercises
 - ▷ Practical Exercises
 - ▷ Submit Exercises
- Review 'Quizzes'

Summary

- Overview Essential Java Language Principles
- Hands-On/Practical
- Today is about becoming comfortable/familiar with Java and the Programming Syntax/Concepts

Questions/Discussion

- Submit Exercise Questions

- 2.1 to 2.12

- Single .zip file with your student number

- Remember to comment your code,
name/student number at the top of files,
separate file for each exercise

- ch2_1.java, ch2_2.java, ...