CS2103T - Week 1 - Topics

W1.1 - OOP: Classes & Objects

W1.2 - OOP: Inheritance

W1.3 - OOP: Polymorphism

W1.4 - Java: Collections

W1.5 - Exception Handling

W1.1 - OOP: Classes & Objects

W1.1a - Paradigms - OOP - Intro - What

- Programming languages divide the world into two parts:
  data and **operations on data**

Paradigm	Language
OOP	Java, C++
Procedural	C
Functional	F#, Haskel, Scala
Logic	Prolog

- Can write procedural code using OO languages

W1.1b - Paradigms - OOP - Objects - What

- Every real world object has:
  1. an **interface** through which other objects can interact with it
  2. an **implementation** that supports the interface but may not be accessible to the other object

W1.1c - Paradigms - OOP - Objects - Objects as abstractions

- **Abstraction**: a technique for dealing with complexity by establishing a level of complexity we are interested in, and suppressing the more complex details below that level

W1·1d - Paradigms - OOP - Objects - Encapsulation of objects

① The packaging aspect

② The information hiding aspect

W1·1e - Paradigms - OOP - Classes - What

W1·1f - C++ to Java - Classes - Defining classes

- Java convention for class names: Pascal Case
- Constructors (new)
- this
- Key exercise: define a Circle class

```
class Circle {
   private int x;
   private int y;
   private double radius;

   Circle() {
      this(0, 0, 0.0)
   }

   Circle(int x, int y, double radius) {
      this.x = x;
      this.y = y;
      this.radius = radius;
   }

   int getArea() {
      return (int) (Math.PI *
Math.pow(this.radius, 2));
   }
}
```

W1.1g - C++ to Java - Classes - Getters and setters

W1.1h - Paradigms - OOP - Classes - Class-level members

W1.1i - C++ to Java - Classes - Class-level members

- static
- final

W1.1j - Paradigms - OOP - Classes - Enumerations

- Enumeration: a fixed set of values that can be considered a data type

W1.1k - C++ to Java - Misc. Topics - Enumerations

- enum
- Convention: all uppercase e.g. FLAG_SUCCESS
- All enums implicitly extend java.lang.Enum. Since a class can only extend one parent, an enum cannot extend anything else
- Java enumerations can have behaviors too

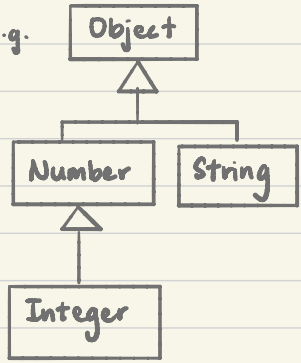
W1.2 - OOP: Inheritance

W1.2a - Paradigms - OOP - Inheritance - What

- Inheritance: allows defining a new class based on an existing class
- Base / parent / super class
- Derived / child / sub / extended class
- Sub-type, is-a relationship e.g. Woman is a Person

- Inheritance hierarchies / trees

e.g.



- Multiple inheritance allowed in Python, C++ but NOT Java, C#

W1·2b - Paradigms - OOP - Inheritance - Overloading

- **Method overloading**: multiple methods with the same name but different type signatures

W1·2c - Paradigms - OOP - Inheritance - Overriding

- **Method overriding**: when a sub-class changes the behaviour inherited from the parent class by re-implementing the method
- Same name, type signature, return type

W1·2d - C++ to Java - Inheritance - Inheritance (Basics)

- super (fields, overridden methods, constructors in superclass)

> Note: If a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass. If the superclass does not have a no-argument constructor, you will get a compile-time error. `Object` does have such a constructor, so if `Object` is the only superclass, there is no problem.

W1.3 - OOP: Polymorphism

W1.3a - Paradigms - OOP - Polymorphism - What

Polymorphism: the ability of different objects to respond, each in its own way, to identical messages

W1.3b - C++ to Java - Inheritance - Polymorphism

- Java is a **strongly-typed** language: the code works with only the object types that it targets

W1.3c - Paradigms - OOP - Inheritance - Abstract classes and methods

- **Abstract class**: cannot be instantiated but can be subclassed
- **Abstract method**: a method signature without implementation
- Class with abstract method = abstract class

W1.3d - C++ to Java - Inheritance - Abstract classes and methods

- abstract
- Cannot instantiate

e.g.

```
abstract class A { ... }

A a; // ok
a = new A(); // compile error
```

ℹ In Java, even **a class that does not have any abstract methods *can* be declared as an abstract class.**

When an abstract class is subclassed, the subclass should provides implementations for all of the abstract methods in its superclass or else the subclass must also be declared abstract.

W1.3e - Paradigms - OOP - Inheritance - Interfaces

- **Interface**: a behavior specification, i.e. a collection of method specifications
- is-a relationship

W1.3f - C++ to Java - Inheritance - Interfaces

- implements

An interface can be used as a type e.g., `DrivableVechile dv = new CarModelX();`.

Interfaces can inherit from other interfaces using the `extends` keyword, similar to a class inheriting another.

Here is an interface named `SelfDrivableVehicle` that inherits the `DrivableVehicle` interface.

```
1 public interface SelfDrivableVehicle extends DrivableVehicle {
2    void goToAutoPilotMode();
3 }
```

Note that the method signatures have no braces and are terminated with a semicolon.

Furthermore, **Java allows multiple inheritance among interfaces.** A Java interface can inherit multiple other interfaces. **A Java class can *implement* multiple interfaces** (and inherit from one class).

Interfaces can also contain constants and static methods.

W1.3g - Paradigms - OOP - Inheritance - Substitutability

- Every instance of a subclass is an instance of the superclass, but not vice-versa

W1.3h - Paradigms - OOP - Inheritance - Dynamic and static binding

- Dynamic (late) binding: a mechanism where method calls in code are resolved at runtime, rather than at compile time

Overridden methods are resolved using dynamic binding, and therefore resolves to the implementation in the actual type of the object.

- Static (early) binding: when a method call is resolved at compile time

In contrast, overloaded methods are resolved using static binding.

W1.3i - Paradigms - OOP - Polymorphism - How

- Polymorphism is achieved by:
  1. Substitutability
  2. Overriding
  3. Dynamic binding

W1.4 - Java. Collections

W1.4a - C++ to Java - Collections - The collections framework

- Contains:
  1. Interfaces : List<E>, Map<K, V>
  2. Implementations : ArrayList<E>, HashMap<K, V>
  3. Algorithms : sort (List<E>)
- Core collection interfaces:
  1. Collection : root of collection hierarchy
  2. Set : cannot contain duplicate elements
  3. List : ordered collection
  4. Queue :
  5. Map : maps keys to values, no duplicate keys, each key can map to at most one value

W1.4b - C++ to Java - Collections - The ArrayList class

- Resizable-array implementation of List interface

W1.4c - C++ to Java - Collections - The HashMap class

- An implementation of the Map interface that stores a collection of key-value pairs

W1.5 - Exception Handling

W1.5a - Implementation - Error Handling - Introduction - What

> Well-written applications include error-handling code that allows them to recover gracefully from unexpected errors. When an error occurs, the application may need to request user intervention, or it may be able to recover on its own. In extreme cases, the application may log the user off or shut down the system. --Microsoft

W1.5b - Implementation - Error Handling - Exception - What

Exceptions: used to deal with 'unusual' but not entirely unexpected situations that the program might encounter at run time

W1.5c - C++ to Java - Exceptions - What are Exceptions?

- Three basic categories of exceptions in Java:

① **Checked exceptions**: conditions a well-written application **should** anticipate and recover from

② **Errors**: conditions **external** to the application, that the application **usually cannot** anticipate or recover from

③ **Runtime exceptions**: conditions **internal** to the application, that the application **usually cannot** anticipate or recover from

W1.5d - Implementation - Error Handling - Exceptions - How

When an error occurs at some point in the execution, the code being executed creates an *exception object* and hands it off to the runtime system. The exception object contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called *throwing* an exception.

After a method throws an exception, the runtime system attempts to find something to handle it in the *call stack*. The runtime system searches the call stack for a method that contains a block of code that can handle the exception. This block of code is called an *exception handler*. The search begins with the method in which the error occurred and proceeds through the call stack in the reverse order in which the methods were called. When an appropriate handler is found, the runtime system passes the exception to the handler. An exception handler is considered appropriate if the type of the exception object thrown matches the type that can be handled by the handler.

the ordered list of methods that had been called to get to the method where the error occurred

The exception handler chosen is said to *catch* the exception. If the runtime system exhaustively searches all the methods on the call stack without finding an appropriate exception handler, the program terminates.

- Advantages of exception handling this way:
  - The ability to propagate error information through the call stack.
  - The separation of code that deals with 'unusual' situations from the code that does the 'usual' work.

W1.5e - C++ to Java - Exceptions - How to use Exceptions?

- try, catch, (finally)
- throw (new Throwable)
- throws (Exception)

W1.5f - Implementation - Error Handling - Exceptions - When

- Only for 'unusual' conditions