

COMP1202 Topics Checklist

Thai Son Hoang

ECS, University of Southampton, U.K.

COMP1202
10th December 2019

Lecture 00 - Starting Out

- ▶ Object-Oriented Programming:
 - ▶ Classes and Objects
- ▶ Logic operators:
 - ▶ ==, !=, &&, ||
 - ▶ >, >=, <, <=

- ▶ From Java code to Program
 - ▶ **Compile**
 - ▶ **Execute**
- ▶ **Java Virtual Machine**
 - ▶ How can Java work on all different platforms
- ▶ Writing a Class in Java
 - ▶ File name convention
 - ▶ **Declaration** of classes, variables, methods
 - ▶ **main** method
- ▶ Conditional **if** .. **else** ... statement
- ▶ Using **brackets** to group statements

- ▶ **Primitive** (**int**, **boolean**, **char**, **float**, ...) vs **Object types**
 - ▶ Defined?
 - ▶ Stored?
 - ▶ Passed?
- ▶ Variable Scope
 - ▶ Member variables
 - ▶ Local variables

- ▶ Method parameters and arguments
- ▶ Passing arguments: Primitive vs. Object types
- ▶ Return type
 - ▶ Using collections to return many things
- ▶ Overloading
 - ▶ Method signature (its name, parameters and order of parameters)
 - ▶ Overloading requires unique signature for each method.

- ▶ Algorithm and characteristics: performance, efficiency, understandability, scalability, reusability, reliability, elegance.
- ▶ Coding style

- ▶ Accessor and Mutator Methods (aka. Getters and Setters)
- ▶ Principle of encapsulation
- ▶ **private** vs **public** keywords
- ▶ Constructors
 - ▶ Naming convention
 - ▶ Does not have a return type
 - ▶ Can be overloaded
 - ▶ Default constructor: No parameter and no behaviour

Lecture 06 - Loops and Arrays

- ▶ Loops: Syntax, condition, loop body, termination
 - ▶ **while** loop
 - ▶ **do . . . while** loop
 - ▶ **for** loop
- ▶ Arrays: Syntax
 - ▶ Declaration
 - ▶ Initialisation
 - ▶ Assignment
 - ▶ Retrieval
- ▶ **ArrayIndexOutOfBoundsException**
- ▶ **for** each loop: Syntax

- ▶ Arrays vs ArrayLists
 - ▶ Declaration
 - ▶ Insertion
 - ▶ Access
 - ▶ Removal
- ▶ Genericity: Syntax
 - ▶ Autoboxing vs unboxing
- ▶ **instanceof** keyword, typecasting
- ▶ Iterators:
 - ▶ **hasNext** () method
 - ▶ **next** () method
 - ▶ Benefit of iterators

- ▶ Implementation vs Interface.
- ▶ Importing: classes, packages, etc.
- ▶ **String** Objects
- ▶ Identity vs. equality (i.e., `==` vs `.equals()` method)
- ▶ **HashMap**: collection that maps a key to a value
 - ▶ `put` method
 - ▶ `get` method

Lecture 09 - IDEs

- ▶ Source code editor
- ▶ Build automation
- ▶ Debugger
 - ▶ Breakpoints

- ▶ Inheritance to avoid code duplication
- ▶ Inheritance and Encapsulation
 - ▶ **public** vs **protected** vs default (no keyword) vs **private**
 - ▶ **super** keyword for calling super-class' constructor.
 - ▶ Superclass constructor rules
- ▶ References and Inheritance
 - ▶ Substitution
- ▶ The `Object` class

- ▶ Overriding methods
- ▶ Which method gets called?
 - ▶ The first method found from the bottom of the inheritance tree is called
- ▶ Dynamic binding: is calling the most specific version of the method, even when the reference has the type of the superclass
- ▶ Polymorphism: Substitution + Overriding + Dynamic binding
- ▶ **super** keyword to access methods in the superclass.

- ▶ Concepts for Code quality:
 - ▶ Duplication: Use methods to write code once and call many times
 - ▶ Coupling: Aims for loose coupling
 - ▶ Cohesion: Aims for high cohesion, i.e. each unit responsible for one single logical task.
- ▶ Responsibility-driven design (RDD): leads to low coupling
- ▶ Thinking ahead and refactoring

- ▶ Error Handling: Advantage and Disadvantage of each approach
 - ▶ Print and default
 - ▶ Error codes
 - ▶ Exceptions: **try** . . . **catch**
- ▶ Debugging:
 - ▶ Syntax errors
 - ▶ Logical errors
- ▶ Testing Strategies
 - ▶ Unit testing: Equivalence Class vs Boundary Value
 - ▶ Manual walkthroughs: Tabulating object state
 - ▶ Verbal walkthroughs: Pair programming
 - ▶ Print statements:
 - ▶ Debuggers

Lecture 14 - Designing Applications

- ▶ Analysis and Design
 - ▶ Noun Verb Analysis
 - ▶ Stepwise Refinement
- ▶ Software Engineering
 - ▶ Documentation
 - ▶ Cooperation
 - ▶ Prototyping
 - ▶ Growth model for iterative development
- ▶ Design Patterns
 - ▶ Pattern structure
 - ▶ Decorator pattern
 - ▶ Singleton pattern
 - ▶ Factory pattern
 - ▶ Observer pattern

Lecture 15 - Abstract Classes and Interfaces

- ▶ **abstract** keyword
 - ▶ **abstract** class
 - ▶ **abstract** method
- ▶ Multiple inheritance vs interfaces
- ▶ **extends** vs **implements** keyword