Contents

Part I

Resources

- UML Distilled textbook by Martin Fowler
- Object-Oriented Software Engineering Practical Software Development using UML and Java (second ed.), Lethbridge, Laganiere
- Head First Object Oriented Analysis and Design, McLaughlin, Pollice, West
- Head First Design Patterns, Freeman, Freeman

Part II

Paradigmes

Chapter 1

Abstraction

There are different meanning of abstractin. one of them is the ability to capture real world entities as classes. Two types of abstractions in Java:

- **interfaces**, used to define expected behaviour.Implementation **is hidden from a client**.
- **abstract classes**, used to define incomplete functionality.

Chapter 2

Inheritance

The ability of subclass to derive members (fields and methods) from ascendands. In java only single parent class is allowed. It is an **'is-a'** relationship. **Derived class inherits all members present in the base class. However not all of them are accessible**. This is ruled by access modifiers used in the base class.

2.1 Accessing Members

It is valid to instantiate an object with a subtype. The instantiated reference variable allows an access to those members (and their variations) which are present in their type. It is still possible to access subtype members using cast:

```
Parent childParent = new Child();
\\ access to a member as it is defined in the Parent class.
childParent.field...

\\ access to a member as it is defined in the Child class
((Child)childParent).field
```

Pay attention to the syntax of above cast!
*** Those method which are overridden are accessible as usual.**
In order to call methods that are not overridden the reference variable used to access the methods (here - *pc*) must be cast to (Child)

2.2 Pitfalls

2.2.1 Constructor Calls an Overridable Method

1. call to constructor in **child class**
2. it calls **parent class constructor** first

3. if there is a call to overridable method it calls **textchild version of the method**

4. **ERROR!** The call **will fail** if the method references some uninitialised variable. **The variable can be initialised only when the control returns to child constructor - in steps which will folow!**

2.3 Liskov Substitution Principle

Whenever an instance of some class is expected in a program, one can suply an instance of subclass of the class

Chapter 3

Encapsulation

Inner details of classes can be hidden by making them private and acceptible through public API only - getters (accessors) and setters (mutators).

Chapter 4

Polymorphism , Method Overriding

'Many forms'. implemented by

- subclass specialisation (*is-a'* relationship
- Liskov substitution principle
- virtual method invocation)

Polymorphism is usually achieved by method overriding. It utilises method dynamic binding.

4.1 Virtual method Invocation

Method calls are dynamically dispatched based on runtime type of the receiver object.

4.2 Method Overloading

Overloading means that two or more methods have the same name but different signature. They **must have different parameters** (number of them and/or types), they **may have different return type and access modifiers** (private, protected, etc) - see Rules.

4.3 Method Overriding

It means that new implementation is provided to an inherited method. This is annotated by @Override. The overridden method in a superclass can be still called when invoked using **super.** keyword.

Rules

- **final, static, private** methods can't be overridden.
- access modifier of overriding method **must not be more restrictive**.
- **no new checked exception** can be thrown
- if return type is a reference type, then it can be original type or any descendant of this type (**covariant return type**).

Private methods can't be overriden because they are excluded from inheritance (not visible from within subclasses).
Static methods reside in static context, they belong to class, not to objects. Therefore they are not inherited, too. Hence, they can't be overriden.
However, they can be **shadowed** - subclass can have static method with the same name, as its parent class. A method call is bind to first method with a proper signature - starting from the class in current context and going up the inheritance tree.

Part III

Design

Chapter 5

Requiremments

5.1 Use Cases