**FIT2099: Exam Excerpts**

**Index**

Contents

[Keeping implementation details of classes private 1](#_Toc10990015)

[Risks of repeated code 1](#_Toc10990016)

[Fixing repeated code 2](#_Toc10990017)

[JavaDocs for private attributes 2](#_Toc10990018)

[Declared Exceptions vs Undeclared Exceptions 2](#_Toc10990019)

[Writing Executable Preconditions with Assertions and Exceptions 3](#_Toc10990020)

[Why not to cope with violated preconditions 3](#_Toc10990021)

[Access control for methods in preconditions 3](#_Toc10990022)

[Benefits of Writing Preconditions before Implementation 3](#_Toc10990023)

[Benefits of Writing Postconditions Before Implementation 3](#_Toc10990024)

[Command-Query Separation, Relation to Design by Contract 3](#_Toc10990025)

[Liskov Substitution Principle and Pre/Post Conditions 3](#_Toc10990026)

[Interface in Java 4](#_Toc10990027)

[Abstract Class in Java 4](#_Toc10990028)

[Interfaces vs Abstract Classes 4](#_Toc10990029)

[Why Reduce Dependencies 4](#_Toc10990030)

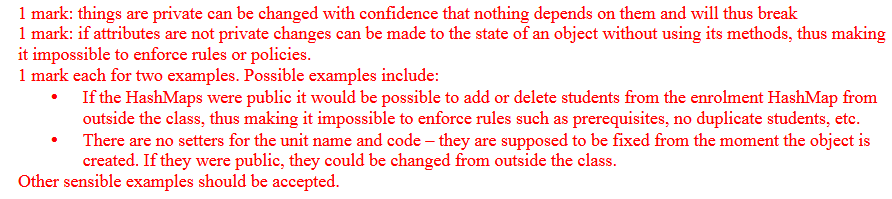
[Encapsulation and Access Modifiers to control dependencies 4](#_Toc10990031)

[What is good abstraction? Abstraction vs Abstract Classes 4](#_Toc10990032)

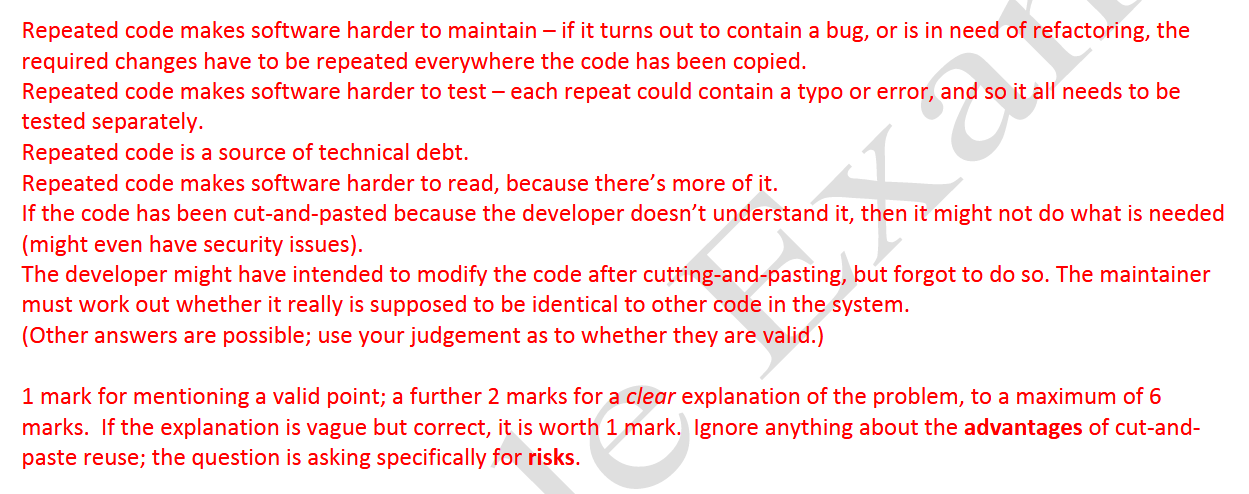
[More Specific Stuff 4](#_Toc10990033)

[Class Diagram and Sequence Diagram Mark Schemes 4](#_Toc10990034)

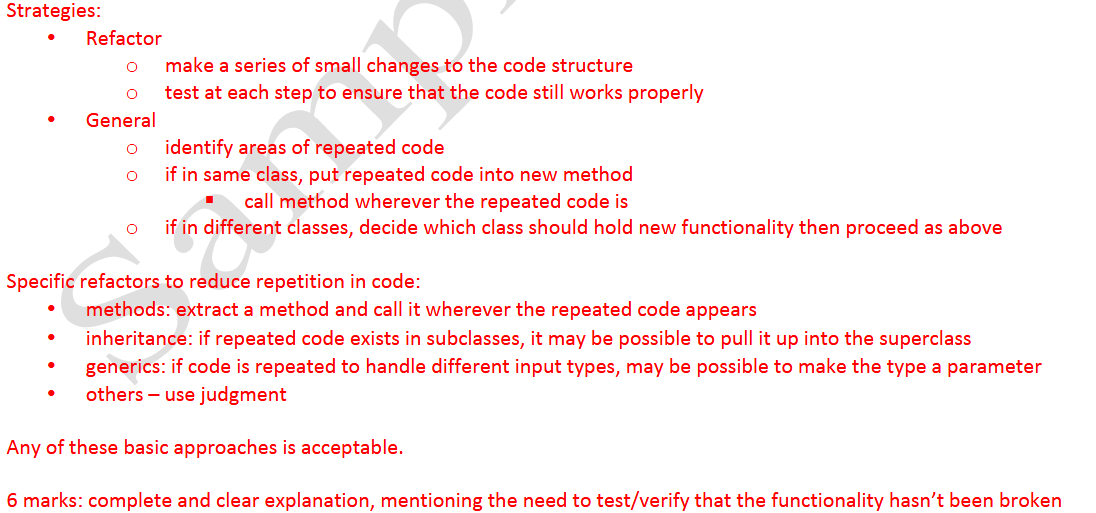
# Keeping implementation details of classes private

****

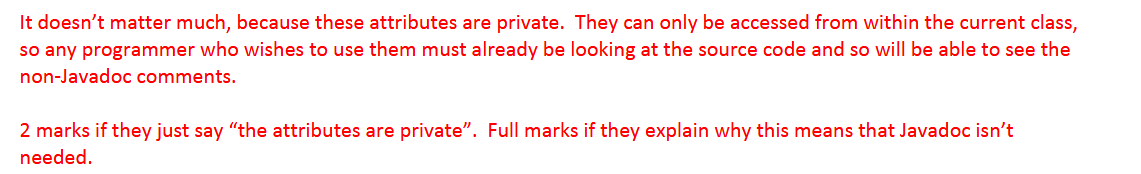
# Risks of repeated code



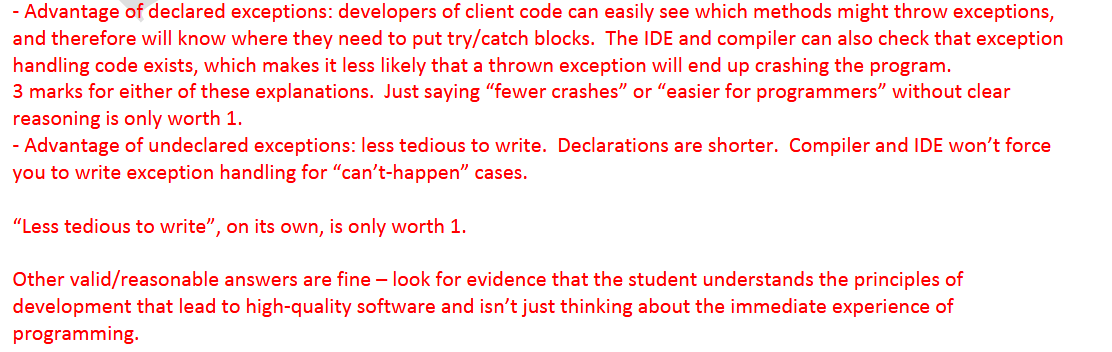
# Fixing repeated code



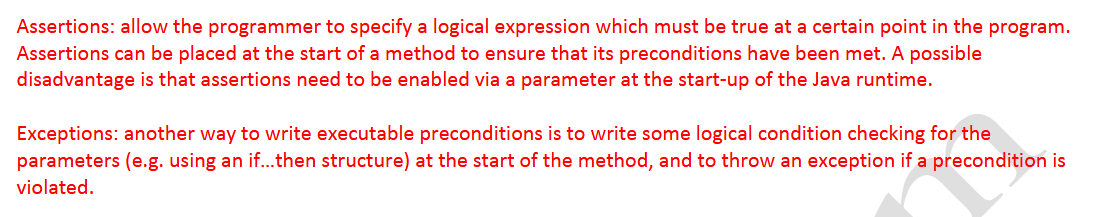
# JavaDocs for private attributes

****

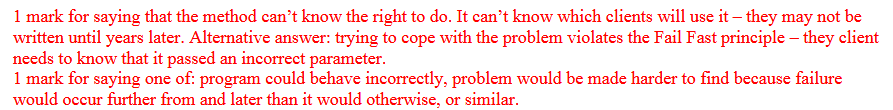
# Declared Exceptions vs Undeclared Exceptions

****

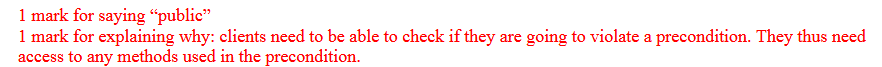
# Writing Executable Preconditions with Assertions and Exceptions

****

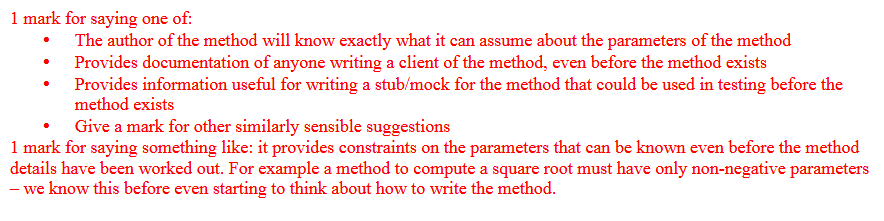
# Why not to cope with violated preconditions



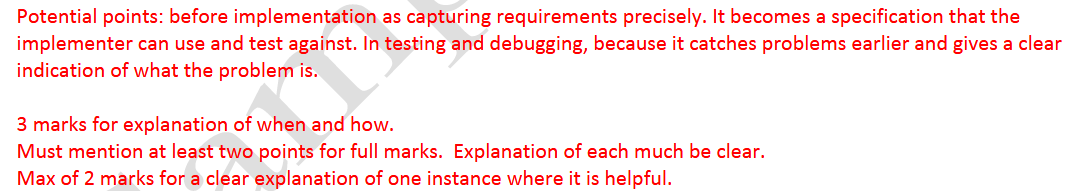
# Access control for methods in preconditions

****

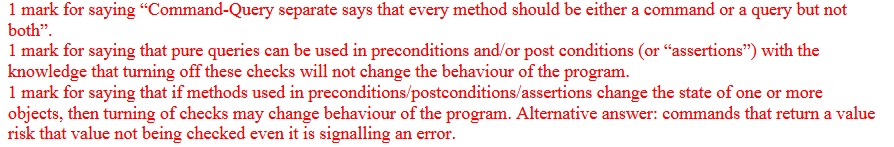
# Benefits of Writing Preconditions before Implementation

****

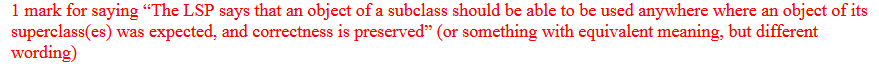
# Benefits of Writing Postconditions Before Implementation

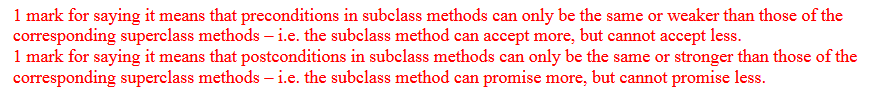
****

# Command-Query Separation, Relation to Design by Contract

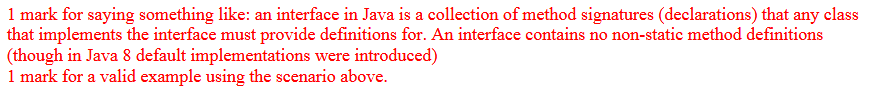


# Liskov Substitution Principle and Pre/Post Conditions

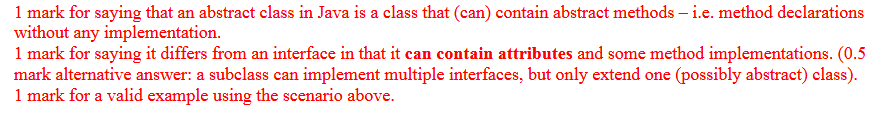
****

****

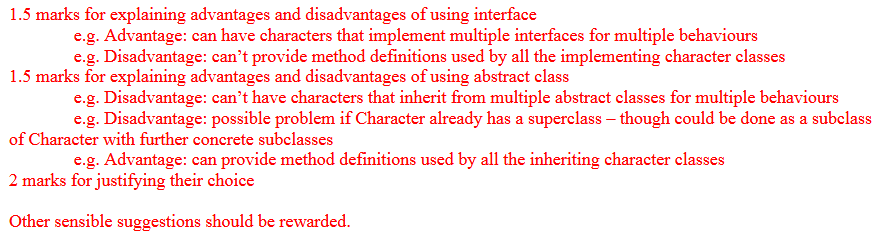
# Interface in Java

****

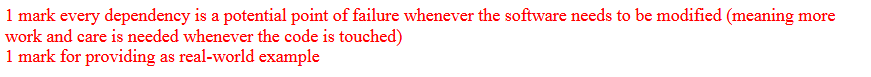
# Abstract Class in Java

****

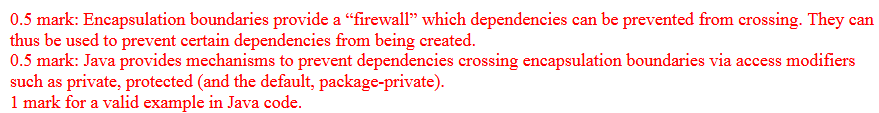
# Interfaces vs Abstract Classes

****

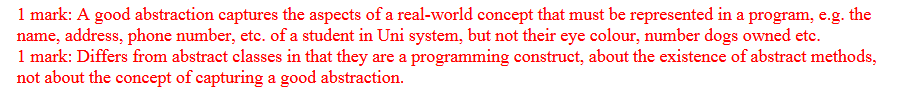
# Why Reduce Dependencies

****

# Encapsulation and Access Modifiers to control dependencies

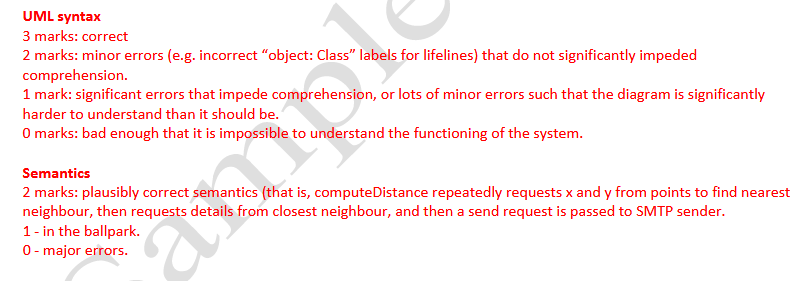
****

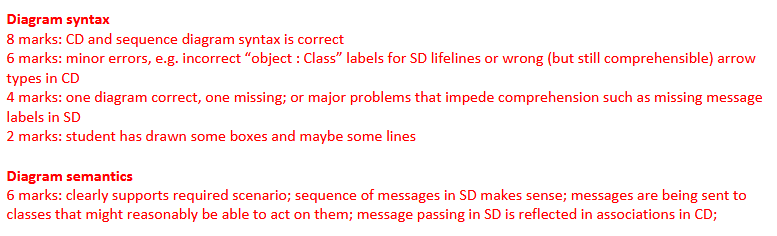
# What is good abstraction? Abstraction vs Abstract Classes

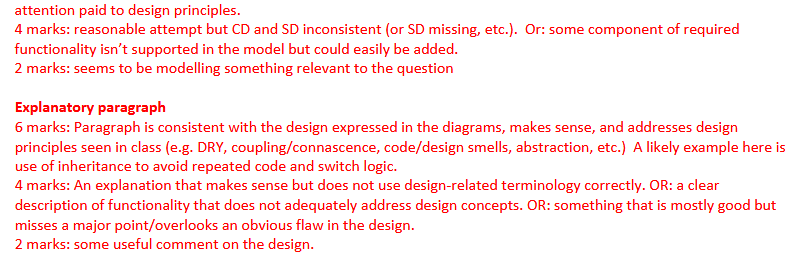
****

# More Specific Stuff

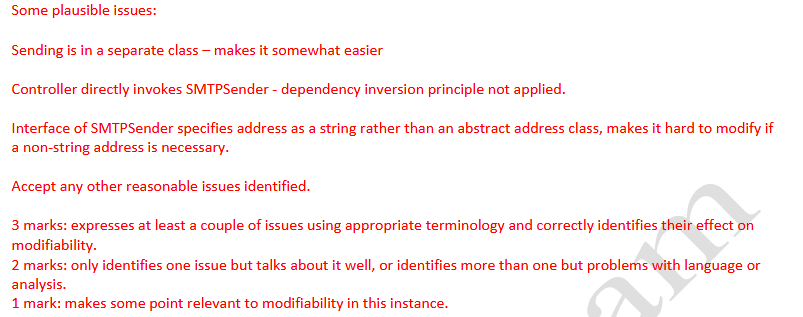
# Class Diagram and Sequence Diagram Mark Schemes



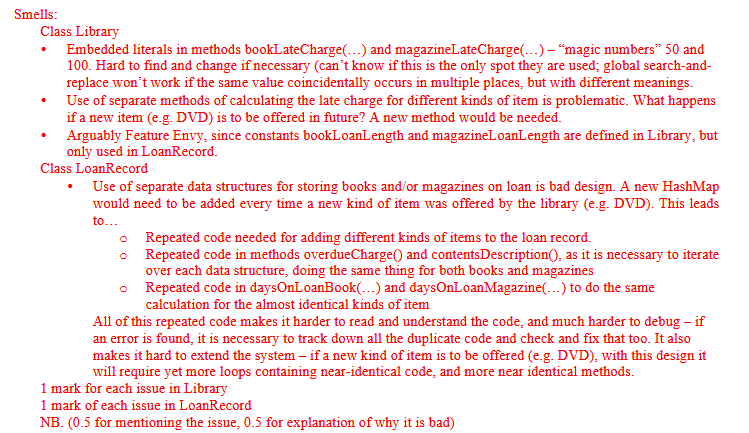


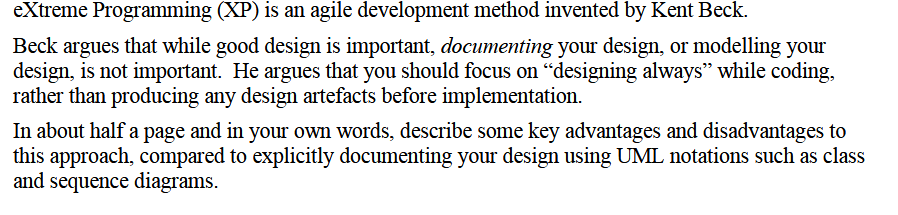


**Ease of Extending a System**

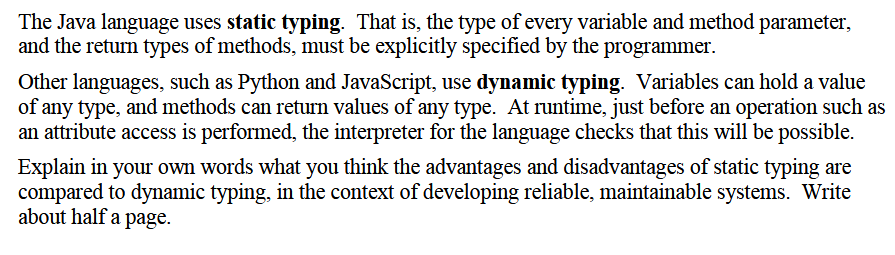
****

**Code Smells**

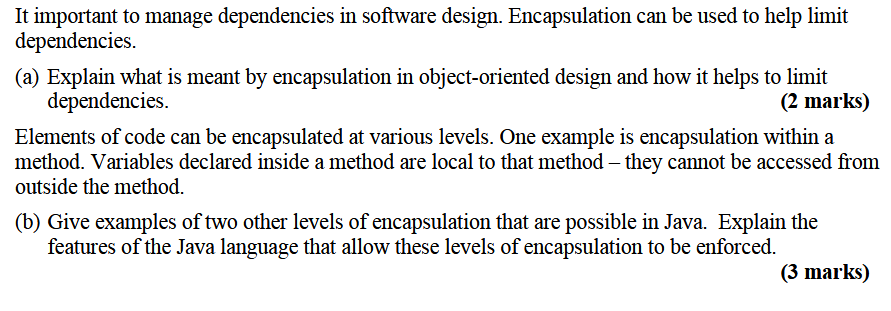
****

****

* Advantages
  + Savings in time (for the short-term) at least
  + Reduces the chance for any redundant design giving more choice to the developers, if done right, the documents should be
* Disadvantages
  + Harder to collaborate with others without proper documentation, people working on the same project would have a harder time getting familiar
  + Harder for others to spot any design smells in your design, may lead to future design flaws
  + Design documents force conscious thinking over what makes a good design, as opposed to this hacking
  + Increases cognitive load

****

* Advantages
  + Type errors are checked at compile time, drastically reducing errors
  + Developer’s have a better track of variable types (reducing cognitive load) and having a lower error rate
* Disadvantages
  + Arguably less tedious for the programmer
  + Less verbose
  + Debug cycle is shorter and less cumbersome due to no compilation step

****

* Encapsulation is the idea of grouping a set of data and operations on the data within a module
* It helps to limit dependencies as it allows things that interact with each other to be grouped together inside an encapsulation boundary and calls that cross the boundary can be properly minimised and controlled
* Within the class and within the package  
  Access modifiers allows methods/classes to be set as public/private/protected allowing developers to control what can be accessed or not  
  Defensive copying by cloning attributes allow for preventing attributes inside the boundary to be modified illegally