EBU6304 – Software Engineering

Analysis and Design

- Analysis
  - Purpose of Analysis
  - Stereotypes of classes
  - Class relationships
- Design
  - Purpose of Design
  - Design principles
  - Design quality
  - Class design

Analysis

What is analysis?

- “A method of studying the nature of something or of determining its essential features and their relations”.
- “A method of exhibiting complex concepts or propositions as compounds or functions of more basic ones”.
- “The evaluation of an activity to identify its desired objectives and determine procedures for efficiently attaining them”.

http://www.dictionary.com

What makes something essential?

We can't take the time to go through all of the possibilities.

No, but we can identify the essential features and relations.

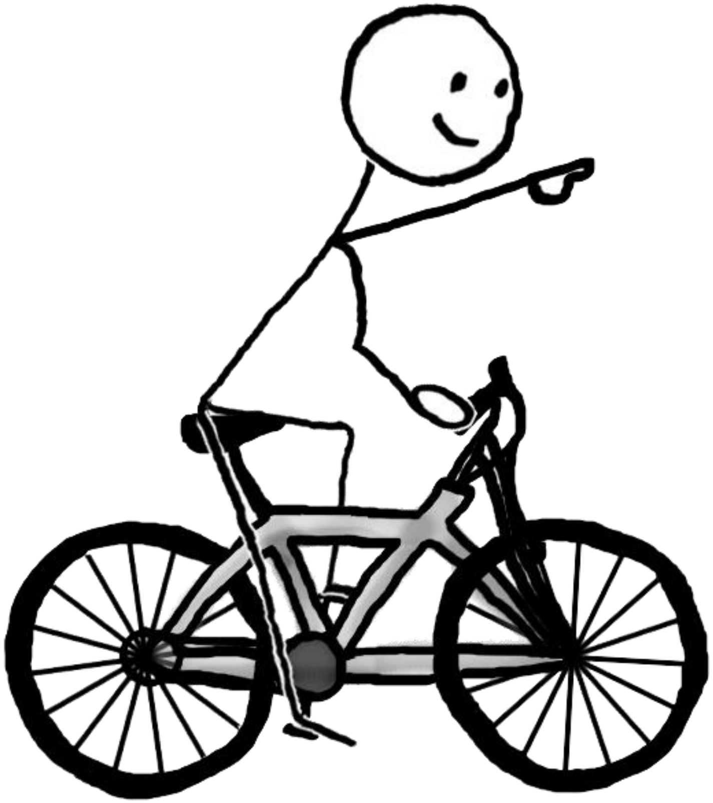
1 Customer won't buy without it.

2 System won't work without it.

3 Many parts of the system depend upon it.

4 …

Why didn’t the customer tell you?

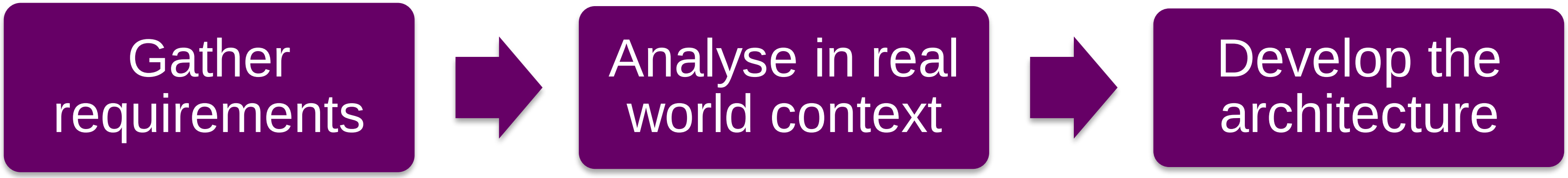


Possible reasons

- They may assume you know about a feature because it’s obvious to them.
- They may not think about some of the special conditions where the feature would be needed.
- They may not know it’s necessary!

Exercise: try to tell someone how you turn left on a bicycle.

Getting to the essentials



Gather requirements → Analyse in real world context → Develop the architecture

- Textual analysis
  - Nouns in requirements and documents.
- Entities and concepts
  - From the application domain.
- Experience
  - Previous systems.

Why “Analyse”

- Focus shifts to developer and system internals.
  - Refining requirements.
- Aim: precise understanding of requirements.
  - Process of structuring requirements:
    - Understand
    - Change
    - Reuse
    - Maintain

What concerns Analysis

- To be used mainly by developers
  - Using the “language” of the developer.
- Provide internal view of the system.
- Conceptual modelling
  - Structured by stereotypical classes and packages.
- Function realisation
  - Outlines how to realise the functionality within the system.

Conceptual modelling

- A conceptual model aims to identify the individual concepts (classes) which exist within a problem domain.
- It should show: (Object Oriented Analysis)
  - Concepts (fundamental classes)
  - Attributes of concepts
  - Operations of concepts (leave details to the design stage)
  - Associations between concepts
- Conceptual models are described using UML Class diagrams.

Class and Object

- Objects are entities that model some concrete or conceptual entity inside the system.
  - A class is an abstraction of an object.
  - Every object belongs to a class, and the class of an object determines its interface (outside world view of the object).
  - The process of creating a new object belonging to a particular class is called instantiating or creating an instance of the class.

Analysis Class

- Analysis classes are conceptual:
  - High level behaviour
  - High level attributes
  - High level relationships and special requirements
- Analysis classes always fit in one of 3 basic stereotypes
  - Entity classes
  - Boundary classes
  - Control classes

Entity classes

- Entity classes
  - Used to model information that is long-lived and persistent
    - Logical data structure
  - Information that the system is dependent on.
  - Store the data and define operations on the data.
    - Search, update, load, save, etc.
  - Data is stored in a database and represented by entity objects in memory.
    - Row in a table <=> object

Entity class example

```
import javax.persistence.*;
import java.io.Serializable;
import java.util.Date;

@Entity(name = "CUSTOMER")     //Name of the entity
public class Customer implements Serializable{
    private long custId;
    private String firstName;
    private String lastName;
    private String street;
    private String appt;
    private String city;
    private String zipCode;
    private String custType;
    private Date updatedTime;
```

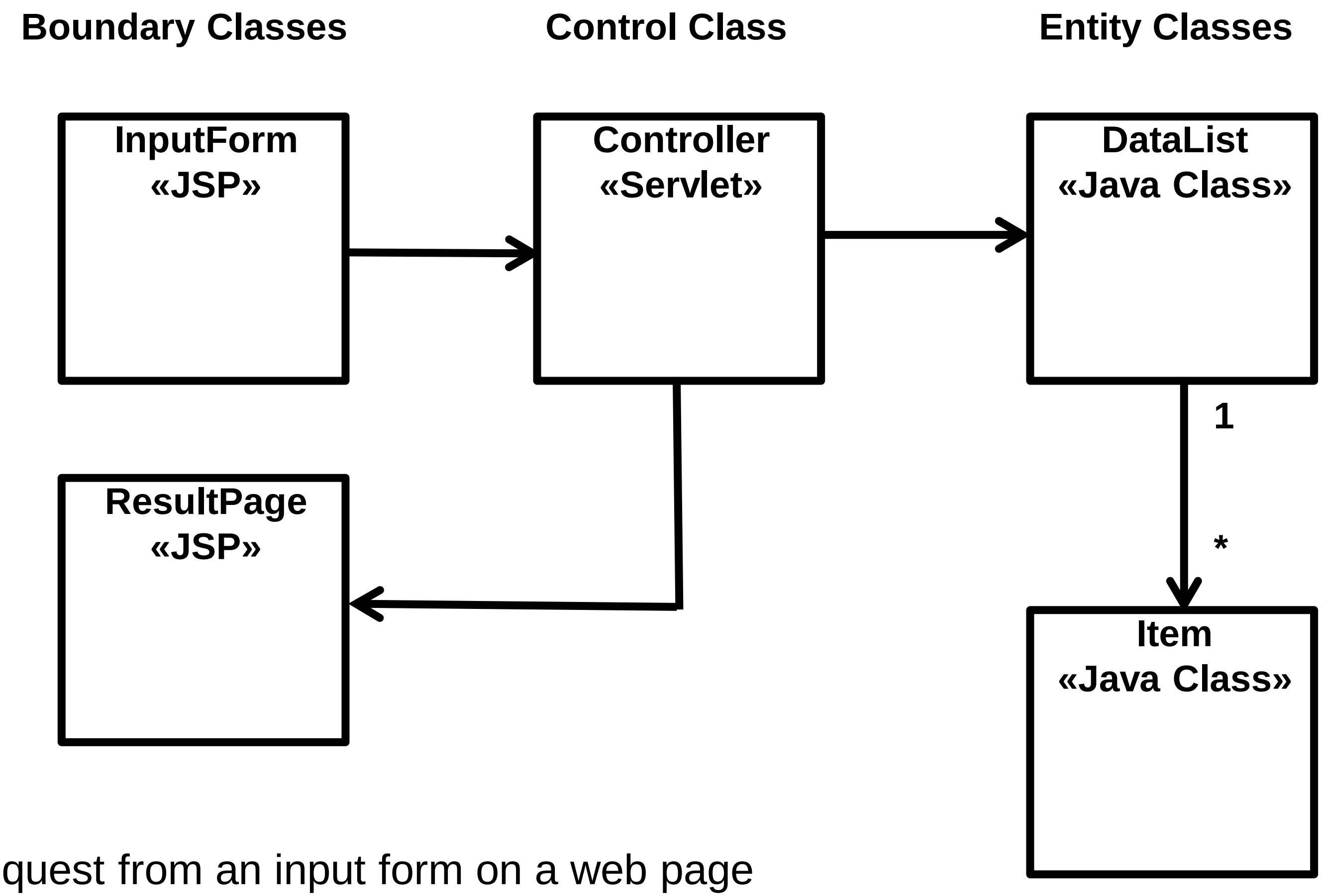
Boundary classes

- Boundary classes
  - Used to model the interaction.
  - Often involve receiving (presenting) information and requests from (and to) users and external systems.
    - Deal with input and output, or connections with rest of system
  - Normally represent abstractions of user/device interface: windows, forms, communication interfaces, printer interfaces, sensors, terminals, etc.

Control classes

- Control classes
  - Used to encapsulate control and coordination of the main actions and control flows.
  - Represent coordination, sequencing, transactions and control of objects.
  - Deal with performing tasks, getting/setting data and coordinating behaviour.

Examples



A request from an input form on a web page
invokes a method call on a controller, which uses
the data class objects to generate a response and
and invokes a boundary class object to display the result.

Attributes

- Attributes are descriptions of a particular data item maintained by each instance of a class.
- Every attribute has a name, a type, and if required a default initial value.
  - During analysis, the attribute name and type can be abstract, for example: *account name*, *string*.
  - During later design, they should have the syntax of the target language, for example: *accountName*, *String*.
- Attributes should be documented with clear, concise definitions.

Operations

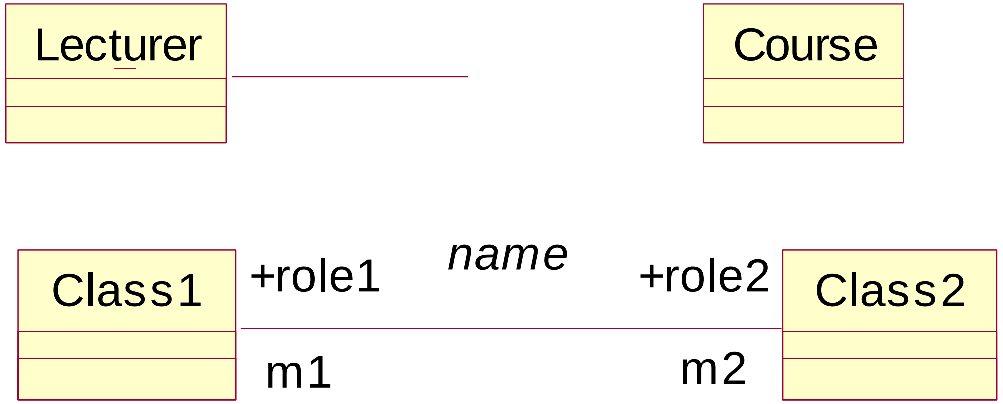
- Operations are abstract specifications of a class's behaviour.
- They have a name, a set of input parameters, and a return type.
  - Details of the functionality of an operation are specified textually.
- An operation should only do one thing:
  - Methods implement operations.
  - Operations should be documented to state the functionality performed by the operation.

Class Relationships

- A system is made up of many classes and objects. Relationships provide the pathway for communication.
- Relationships
  - Association
  - Inheritance
    - Generalisation
    - Specialisation

Association

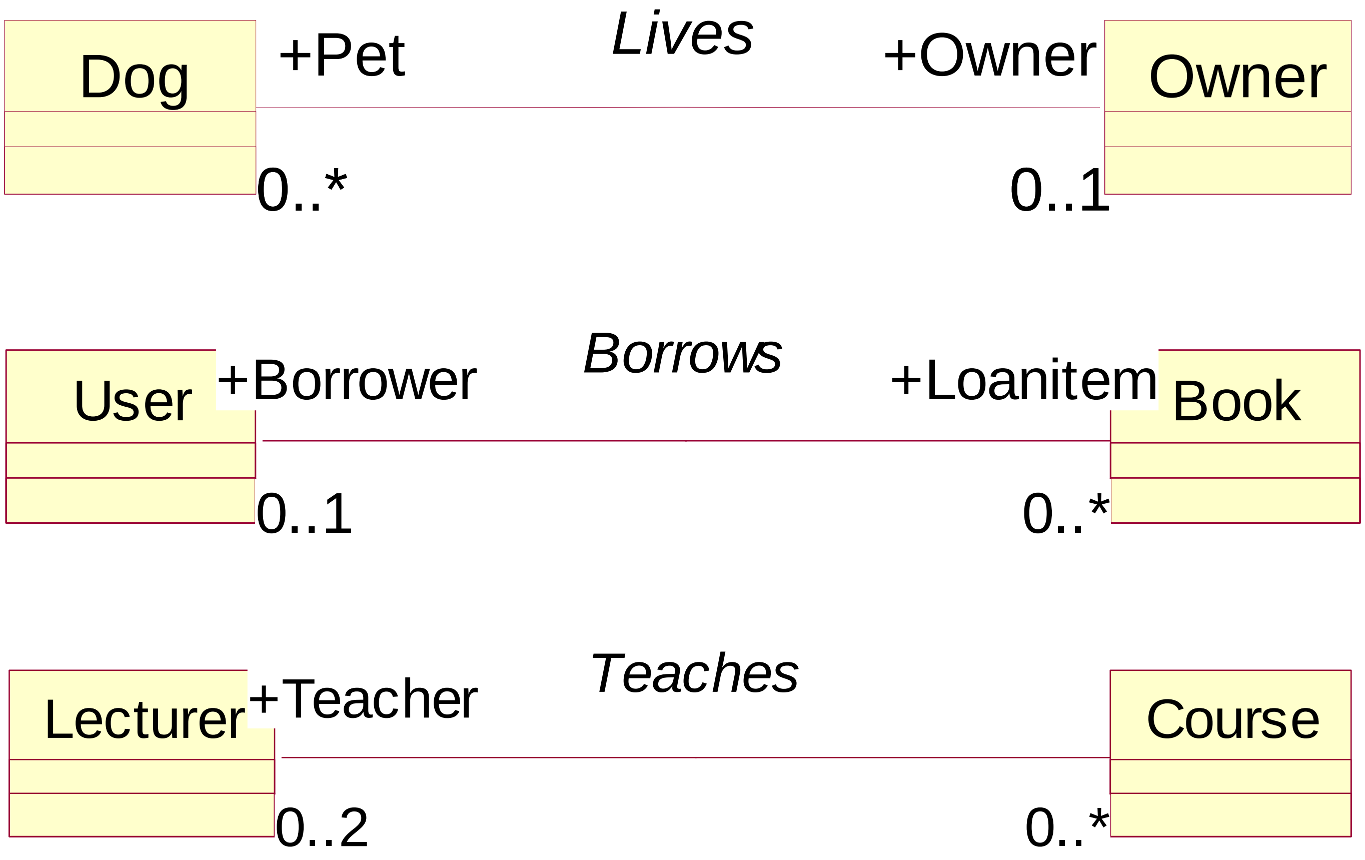
- An association is a bidirectional semantic connection between classes:
  - Data may flow in either direction.
- An association means there is a link between objects.



Association

- Associations have:
  - a name, the meaning;
  - a role, which describes the role the instances of the associated class play in the relationship;
  - a multiplicity which states how many instances of a class at one role end can be associated with an instance of another class at the other role end.

Association Examples

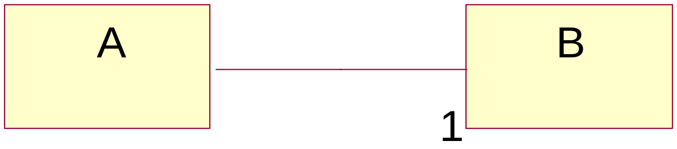


Association Multiplicity

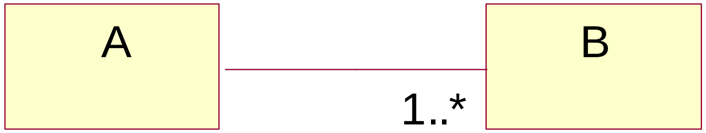
- Multiplicity indicators
  - 1 Exactly one
  - 0..* Zero or more
  - 1..* One or more
  - 0..1 Zero or one
  - 5..8 Specific range (5, 6, 7, 8)
  - 4..7, 9 Combination range (4,5,6,7 or 9)

Association Multiplicity

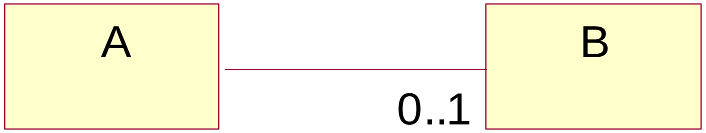
- Examples:



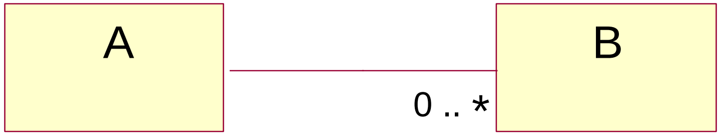
An A is associated
with exactly one B.



An A is associated
with one or more B.



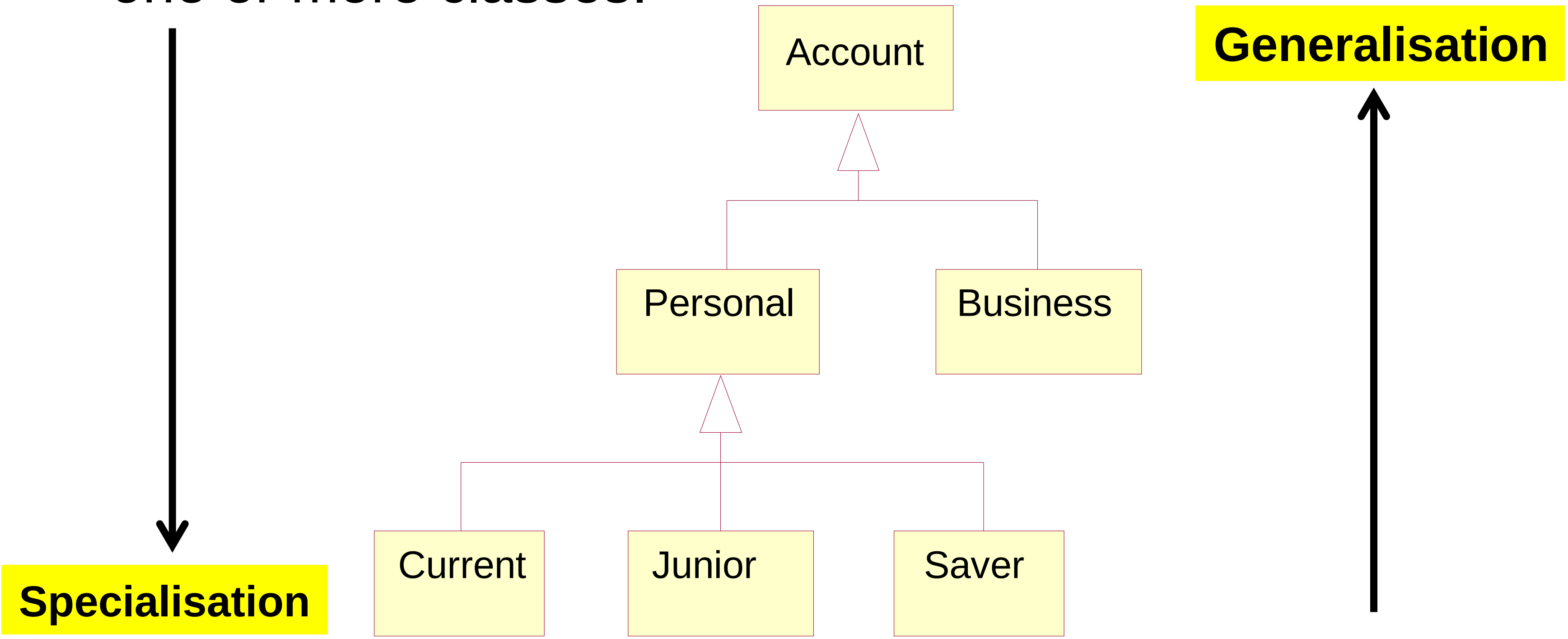
An A is associated
with zero or one B.



An A is associated
with zero or many B.

Inheritance

- Inheritance defines a relationship among classes where one class shares the attribute(s) and/or operation(s) of one or more classes.

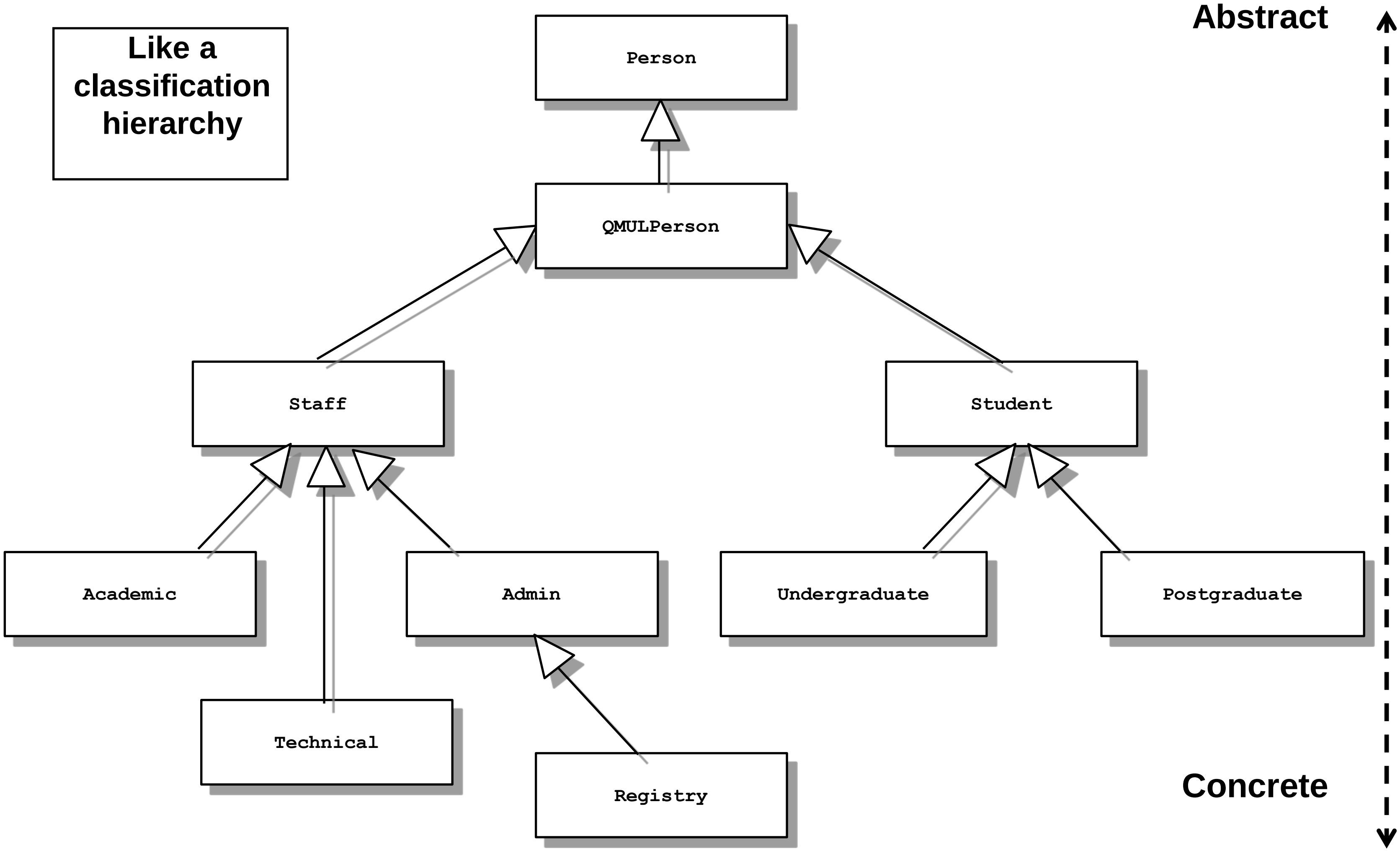


Inheritance

- “is-a”, “kind-of” hierarchy
- A subclass will inherit all attributes, operations, relationships defined in any of its superclasses.
- Subclass may be augmented with additional attributes and operations.
- Subclass can override attributes and operations.
- The key to reuse.

More examples

Like a classification hierarchy



Analysis steps

Activities:

1. Identify Entity, Boundary and Control classes
2. Identify class relationships
3. A conceptual class diagram
4. Identify attributes for each entity class
5. Add constraints

Design

What is design?

I thought we were doing design already? Is there really a big difference between analysis and design? When can we get to the coding?

Blueprint

Plan

Structure

The purpose for something

Style

I looked up “design” on the Web and found a lot of different definitions. Which is right?

Common design characteristics

- Designs have a purpose
  - They describe HOW something will work.
- Designs have enough information so that someone can implement them.
- There are different styles of design
  - Like different types of house architectures.
- Designs can be expressed at different levels of detail
  - A dog house needs less detail than a skyscraper.

Our definition of “design”

- Software design is the process of planning how to solve a problem through software.
- A software design contains enough information for a development team to implement the solution. It is the embodiment of the plan
  (i.e. the *blueprint for the software solution*).

Role of Design

- Design transforms the analysis model into a design model that serves as a blueprint for software construction.
- At this point, consideration needs to be taken for the non-functional requirements e.g.
  - The programming language chosen
  - Operating systems
  - Databases
  - User-interfaces
- During the design phase: break down the overall task.
- Create a ‘skeleton’ of the system that the implementation can easily fit into.

Design Quality Guidelines

- A good software design should:
  - Meet the requirements
  - Be well structured: exhibit an architecture
  - Be modular
  - Contain distinct representations of data, architecture, interfaces, and components
  - Be maintainable
  - Be traceable
  - Be well documented: represented using a notation that effectively communicates its meaning
  - Be efficient (when implemented)
  - Be error free

Fundamental Concepts

- Abstraction: data, procedure, control
- Architecture: overall structure of the software
- Patterns: a proven design solution
- Modularity: compartmentalization
- Information hiding: encapsulation
- Functional independence: coupling and cohesion
- Refinement: elaboration of detail for all abstractions
- Refactoring: a reorganization technique that simplifies the design

Abstraction

Abstract class

- Defines behavior
- Can have implementation code
- Cannot be instantiated
- A class can inherit from a single abstract class
  - Unless the language supports multiple inheritance

IS-A

Interface

- Defines behavior
  - Contract
- Cannot be instantiated
- A class can implement multiple interfaces
  - In languages that support interfaces

CAN-DO

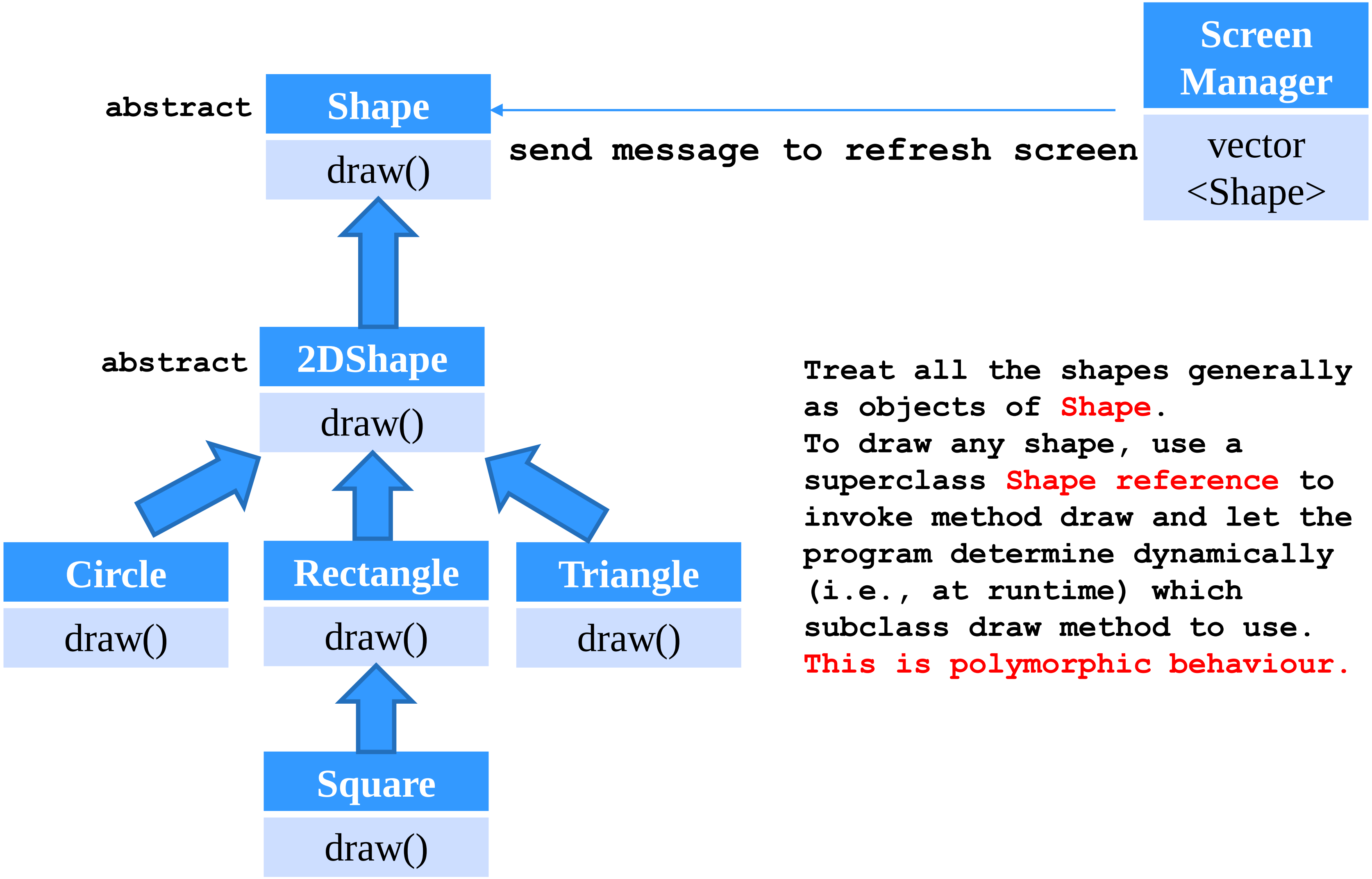
Abstraction

- How do I know when to use an interface and when to use an abstract class?
  - If (almost) all classes implementing the behavior would have the same code, then you can use an abstract class to implement it.

Program to interfaces

Avoid repeating code with abstract classes

Abstraction



Treat all the shapes generally as objects of Shape.
To draw any shape, use a superclass Shape reference to invoke method draw and let the program determine dynamically (i.e., at runtime) which subclass draw method to use.
This is polymorphic behaviour.

Encapsulation

- Restricting of direct access to some of an object's components
  - Information hiding
- Bundling of data with the methods that operate on that data
  - Implementations of abstract data types

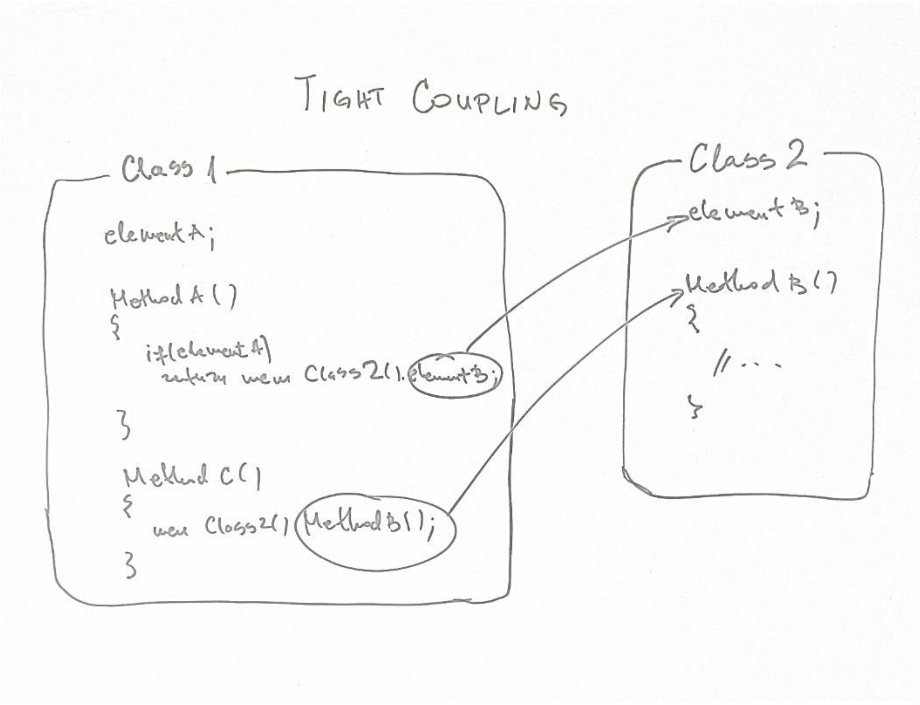
Modularity

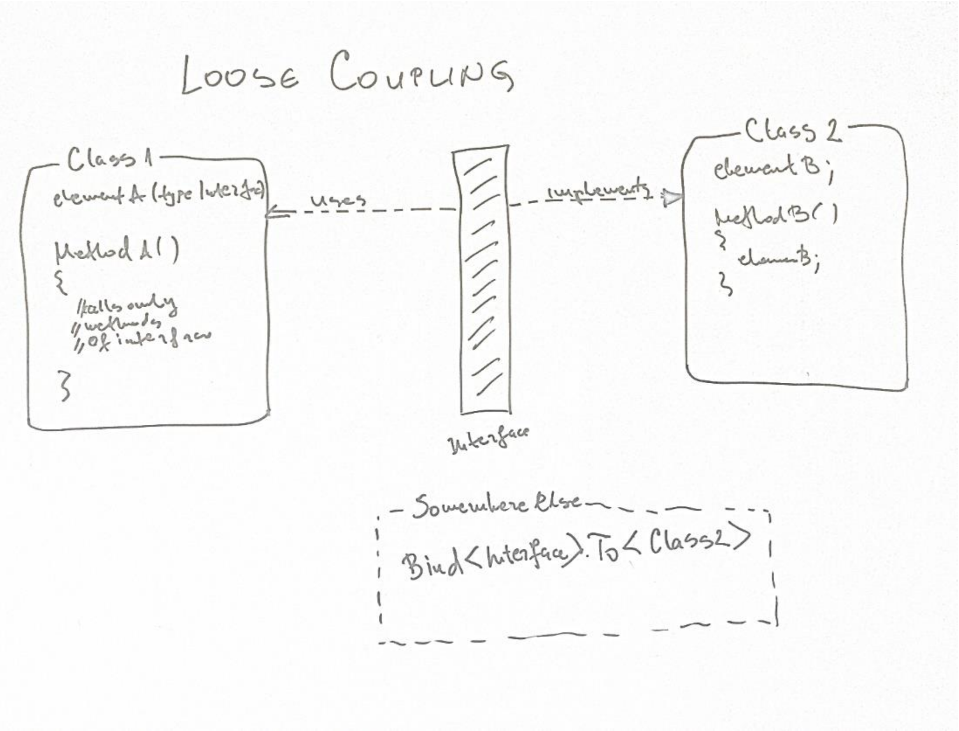
- Separate the functionality of a program into independent, interchangeable modules
- Each module contains everything necessary to execute only one aspect of the desired functionality.
- A module interface expresses the elements that are provided and required by the module.
- The elements defined in the interface are detectable by other modules.

Coupling

- The number of dependencies between subsystems.
- Indicates strengths of interconnections
  - Tight: relatively dependent. Modifications to one is likely to have impact on others.
  - Loose: relatively independent. Modifications to one will have little impact on others.
- Ideally, subsystems are as loosely coupled as reasonable …
  - to minimise the impact on errors or future change.

Tight/Loose coupling



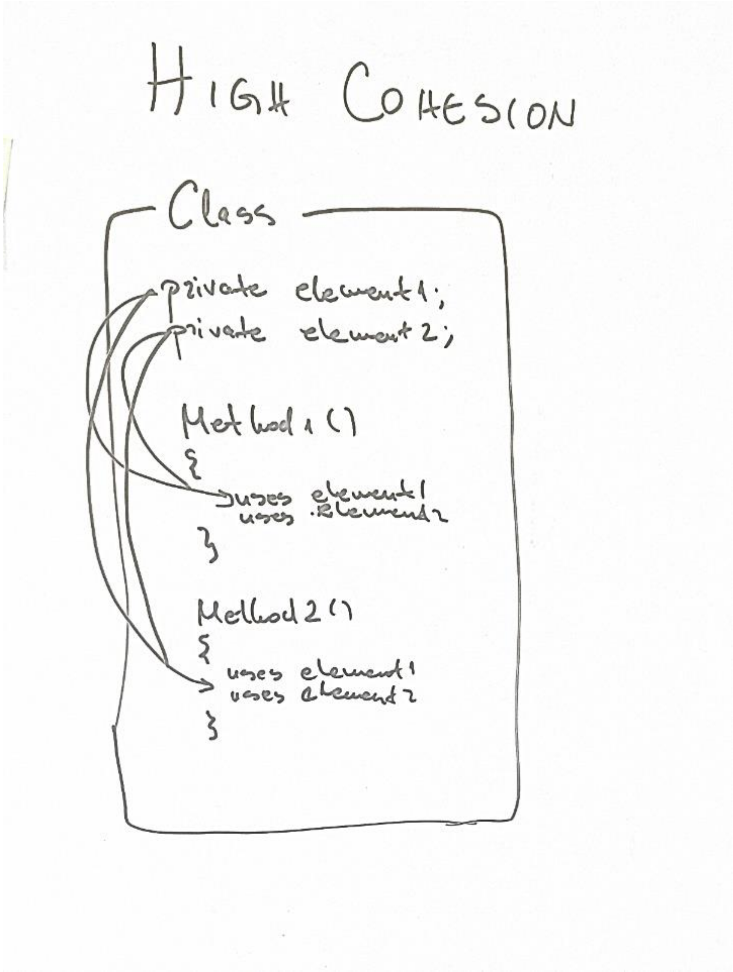


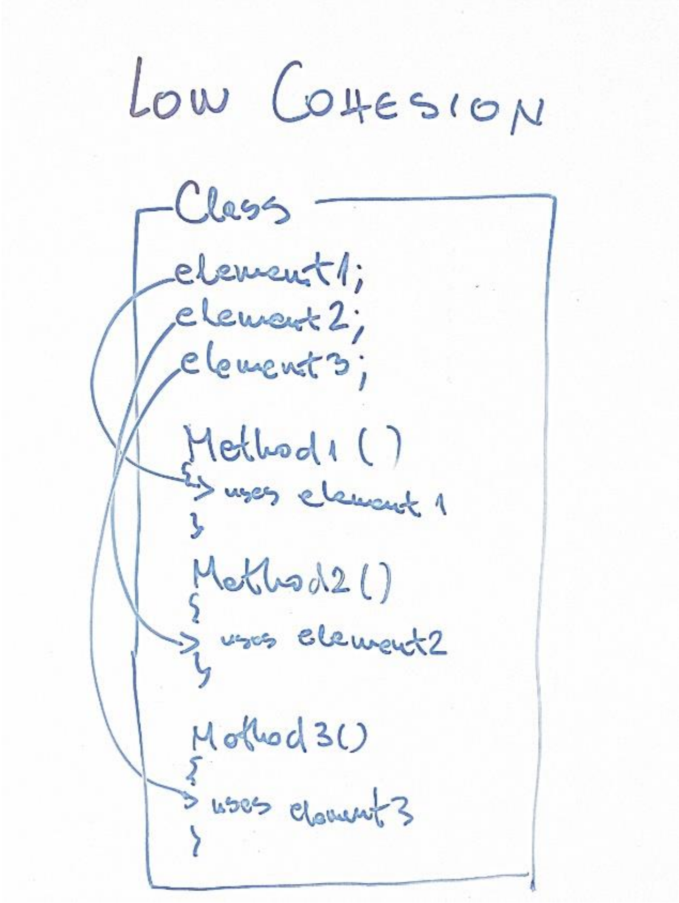
https://thebojan.ninja/2015/04/08/high-cohesion-loose-coupling/

Cohesion

- The number of dependencies within a subsystem.
- A measure of the level of functional integration within a module.
  - High: objects are related to each other and perform similar tasks.
  - Low: unrelated objects.
- Ideally, a subsystem should have high cohesion.
  - All parts of the component should contribute to its logical function.
  - If it is necessary to change the system, then everything to do with the component is encapsulated in one place.

High/Low cohesion



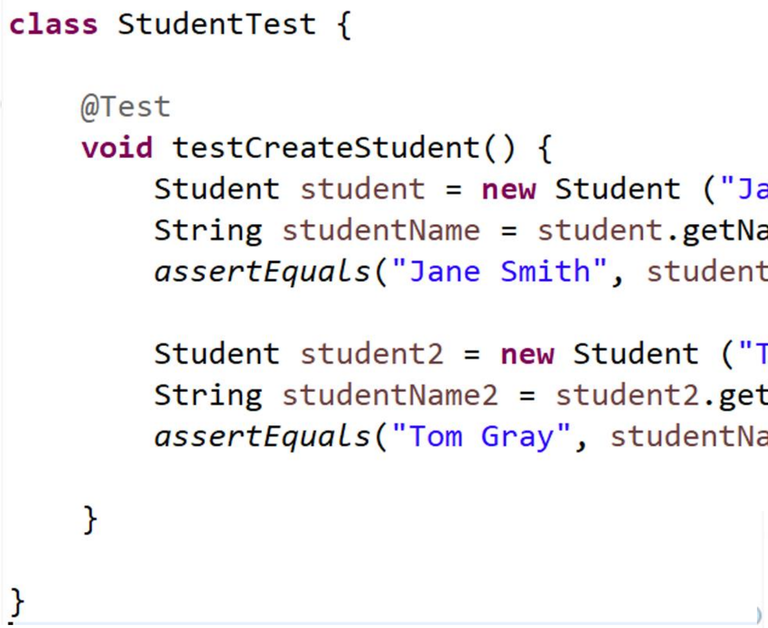


https://thebojan.ninja/2015/04/08/high-cohesion-loose-coupling/

Refactoring

- First: get the code to work.
- Second: ensure that the code stays clean.
  - No duplicate code in the system
  - The code is clean and expressive, clearly stating the intent of the code
- Refactoring
  - Frequently review/change code, without changing its external behaviour
  - Refactoring is intended to improve nonfunctional attributes of the software

Refactoring



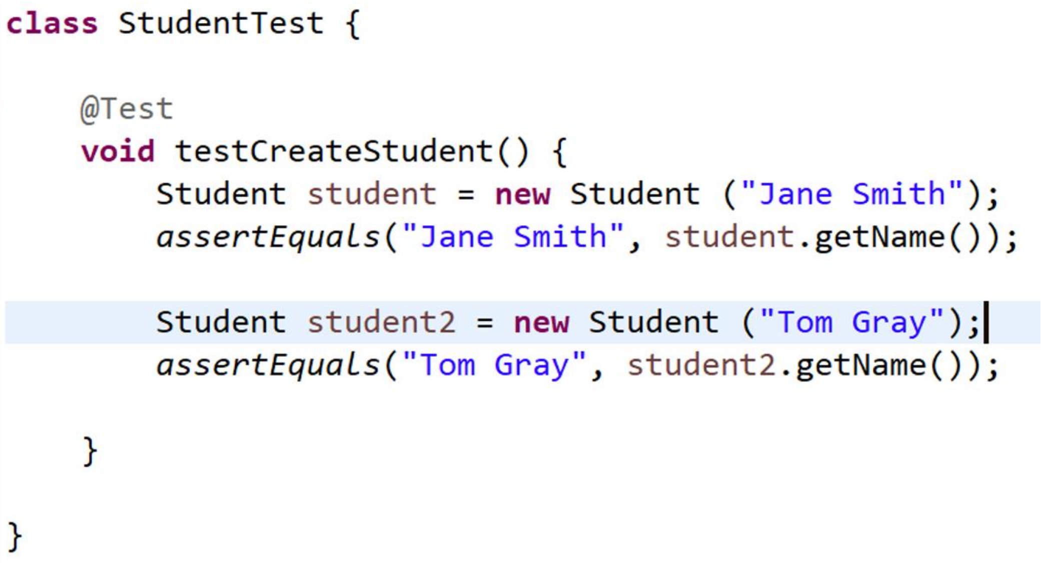
```
class StudentTest {

    @Test
    void testCreateStudent() {
        Student student = new Student ("Jane Smith");
        String studentName = student.getName();
        assertEquals("Jane Smith", studentName);

        Student student2 = new Student ("Tom Gray");
        String studentName2 = student2.getName();
        assertEquals("Tom Gray", studentName2);

    }

}
```



```
class StudentTest {

    @Test
    void testCreateStudent() {
        Student student = new Student ("Jane Smith");
        assertEquals("Jane Smith", student.getName());

        Student student2 = new Student ("Tom Gray");
        assertEquals("Tom Gray", student2.getName());

    }

}
```

Advantages of Object Oriented Design

- Easier maintenance:
  - – Objects are independent.
  - – Objects may be understood as stand-alone entities.
- Objects are potentially reusable components:
  - – Reuse previous developed objects
  - – Standard object
  - – Inheritance
- For some systems, there may be an obvious mapping from real world entities to system objects.

`More will be introduced in later lectures in week 3 and 4`

Design steps

Activities:

1. Based on the conceptual class diagram produced from the Analysis stage.
2. Identifying Class Relationships: Associations / Generalisations
3. Identify operations
4. Describing methods
5. Captures implementation requirements.
6. Produce detailed design class diagram.

Summary

- The aim of analysis is to precise understanding of requirements
- A conceptual model aims to identify the individual classes.
- Analysis classes always fit in one of 3 basic stereotypes: Entity classes, Boundary classes, Control classes
- Relationships: Association, Inheritance
- Design transforms the analysis model into a design model that serves as a blueprint for software construction
- Fundamental Concepts

References

- Chapter 4, 5 – “Head First Object Oriented Analysis & Design” textbook by Brett McLaughlin *et al*
- Chapters 6, 7 – “Software Engineering” textbook by Ian Sommerville