Bjørn Sand Jensen
bjorn.jensen@glasgow.ac.uk

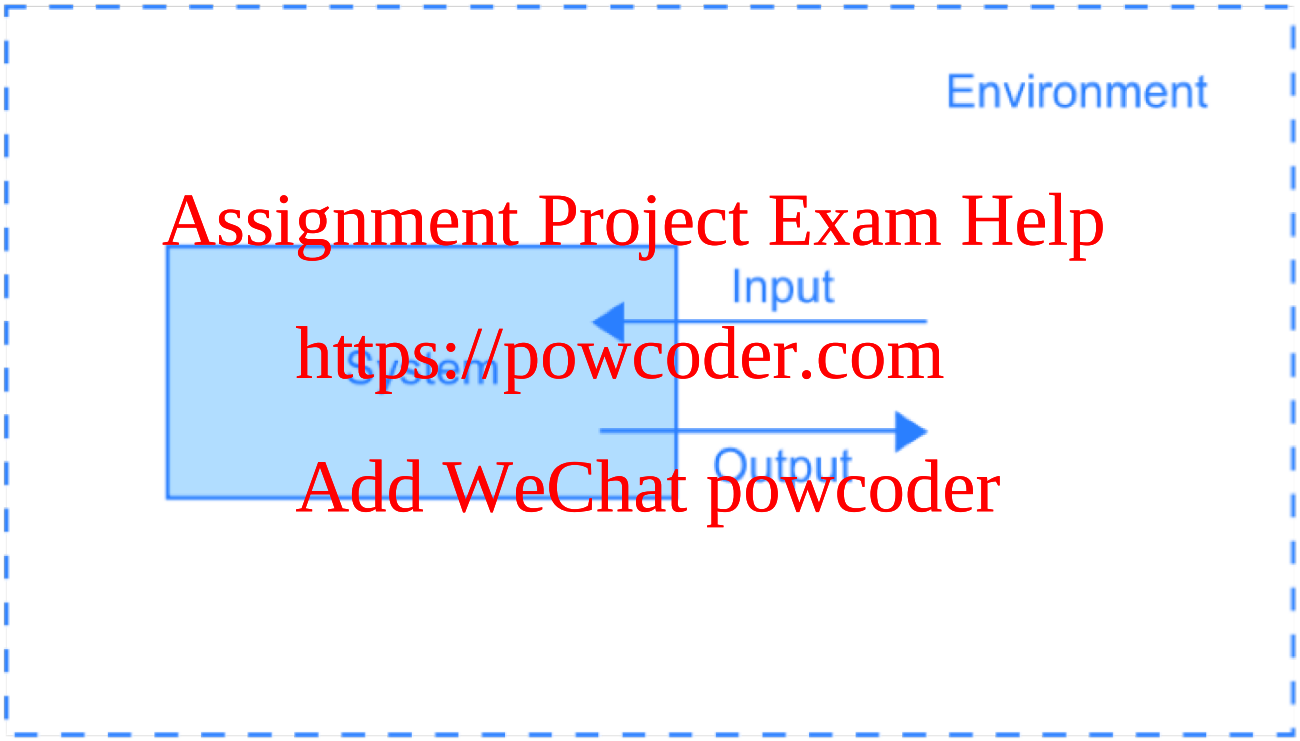
Lecture 1

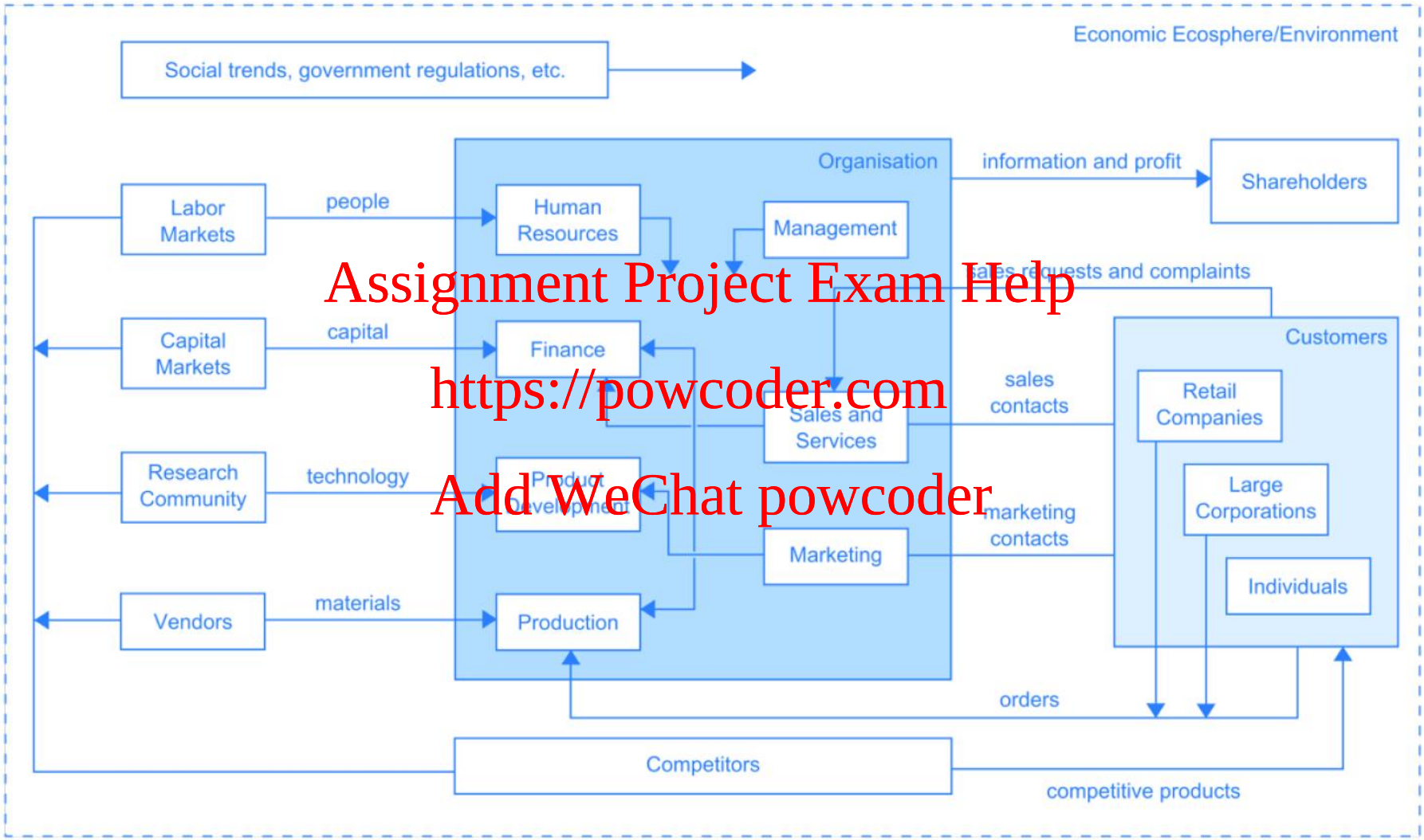
(Slides adopted from SE(M) 2016 by Dr Julie Williamson and slides from the OOSE book repository)

Today's Plan

- 10:05-10:55:
  - What is software engineering?
  - Housekeeping (and how to pass this class)
- Break
- 11:05-11:55:
  - Software development process
  - Requirements (mostly review)
- Long break
- 14:05-14:55: Object Orientation – important concepts (mostly review) [Sir Alexander Stone 208, no lab!]

What is Software Engineering ?





Software Development is Hard

- Software is intangible

Software Development is Hard

- Software is Complex

Software Development is Hard

- Software Fails

- http://spectrum.ieee.org/computing/software/why-software-fails

Software Development is Hard

- Software is Malleable

Characteristics of Good Software

- Easy of use
- Modularity
- Modifiability
- Efficient
- Correct
- Maintainability
- Understandability
- Reusable
- Portable
- Security
- ... etc

What is Software Engineering?

- "Software engineering is an engineering discipline that is concerned with all aspects of software production" (Sommerville)
- "A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers." (Ghezzi, Jazayeri, Mandrioli)
- "The application of a systematic, disciplined, quantifiable approach to the development, operation, maintenance of software; that is, the application of engineering to software…." IEEE
- **"The process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints**

Why is Software Engineering Important?

Software Engineering is Design!

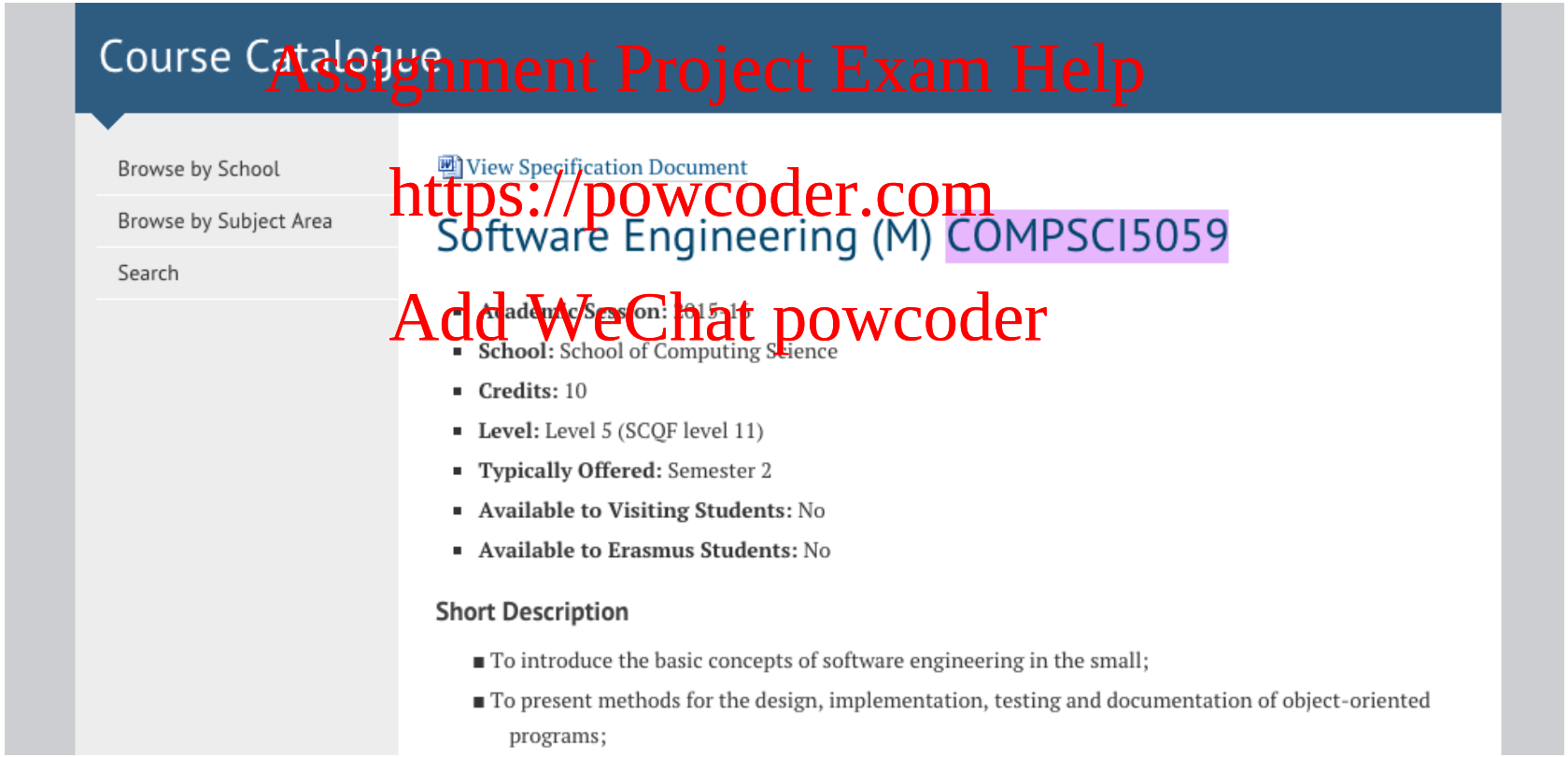
- Very much related to the **design** of the internal structure software to meet the external requirements.

- This is the fun aspect of software development !

Course Aims

- *To introduce the basic concepts of software engineering in the small;*
- *To present methods for the design, implementation, testing and documentation of object-oriented programs;*
- *To develop program comprehension and design skills by studying and extending existing programs.*

Intended Learning Outcomes and Course Objectives



Intended Learning Objectives

- *1: Develop clear, concise, and sufficiently formal documentation of extensions to an existing system, based on the true needs of users and other stakeholders;*
- *2: Apply design principles and patterns while designing and implementing simple systems based on reusable technology;*
- *3: Produce documentation appropriate for programs developed in practical exercises;*
- *4: Create UML class diagrams which model aspects of the domain and the software architecture;*
- *5: Carry out testing of programs and apply simple measurement techniques to software;*
- *6: Discuss the breadth of software engineering practices.*

		Topic	Who	Assigned Reading (OOSE, unless otherwise indicated)	Morning Session 10:00-12:00 MB420	Afternoon Session 14:00-15:00 BO1028, if not otherwise indicated	Assessed coursework
W17	8/1	Introduction and review	BSJ	Ch 1+2 (review Ch 4)	Lec 1.1	Lec 1.2 (in SAS208) Lab 1: Review Eclipse (review outside of class)	(PI1/Q1, not assessed; demo only)
W18	15/1	Modelling and UML	BSJ	Ch 5	Lec 2	Lab 2: Design using UML	PI2/Q2
	19/1						AE1 Handout
W19	22/1	Design Patterns	KY	Ch 6	Lec 3	Lab 2: Patterns	PI3/Q3
W20	29/1	Design Patterns	BSJ	Ch 6	Lec 4	Lab 3: Patterns	PI4/Q4
W21	5/2	Design Principles and Architecture	BSJ	Ch 9	Lec 5	Lab 4: Design Principles	PI5/Q5
W22	12/2	Design Principles and Architecture (incl. documentation)	BSJ	Ch 9	Lec 6	Lab 5: Design Principles and Documentation	PI6/Q6
	16/2						AE1 Deadline & AE2 Handout
W23	19/2	Testing	KY	Ch 10	Lec 7	Lab 6: Test	PI7/Q7
W24	26/2	Testing	KY	Ch 10 +TBD	Lec 8	Lab 7: Test	PI8/Q8
W25	5/3	Continuous Integration	KY		Lec 9	Lab 8: Continuous integeration	PI9/Q9
W26	12/3	Revision	KY (BSJ)	Revision	Lec 10	Lab revision	PI10/Q10
	16/3						AE2 Deadline

Course Details

- The syllabus is available on Moodle
- http://moodle.gla.ac.uk/course/view.php?id=2780

Primary Textbook & Videos

- Object Oriented Software Engineering – Lethbridge and Laganiere (available as FREE e-book)

- http://www.site.uottawa.ca/school/research/lloseng/

- Classical one-way lectures available from:
  - http://www.site.uottawa.ca/school/research/lloseng/supportMaterial/videos/
  - Note: Our aim is not to repeat the recorded lectures but to get you involved and help you to reflect on the material during lectures.

Venues

- Lectures (w/quizzes and exercises):
  - Week 1-10: Monday 10:00-12:00 Main Building 420
  - Week 1-10: Monday 13:00-14:00 Sir Alexander Stone 208
- Labs
  - Week 2 onwards:
    - Monday 13:00-14:00 Boyd Orr Building 1028

      OR
    - Monday 14:00-15:00 Boyd Orr Building 1028

Lab Scheduling

Monday 13:00-14:00 Boyd Orr Building 1028

OR

Monday 14:00-15:00 Boyd Orr Building 1028

?

Please indicate your preference via Moodle:

http://moodle2.gla.ac.uk/mod/choice/view.php?id=819064

Who are we?

- Lecturer: Bjørn Sand Jensen (aka Bjørn / Bjorn)
  - Office: SAWB 306
  - Email: Bjorn.Jensen@glasgow.ac.uk
- Lecturer: Ke Yuan
  - Office: SAWB 205
  - Email: Ke.Yuan@glasgow.ac.uk

Who are we?

- What we (also) spend our time on…
  - Research, rarely on SE aspects per say.
  - We are both Machine Learning researchers. We do Machine Learning, AI and Signal/Image Processing in biology, cancer, audio, images, text, human behaviour….
  - We regularly **reuse**, **extend**, **develop** and **apply** scientific software
- My personal experience with software engineering…

Who are you?

Course assessment

- Exam, 70%
- Course-work, 30%

Coursework

- 30% of your total mark
- We expect well written well presented work.
- Coursework must have a cover sheet
- Hand in times are at 4:30 on the due date
- No questions will be accepted two days before assignments are due

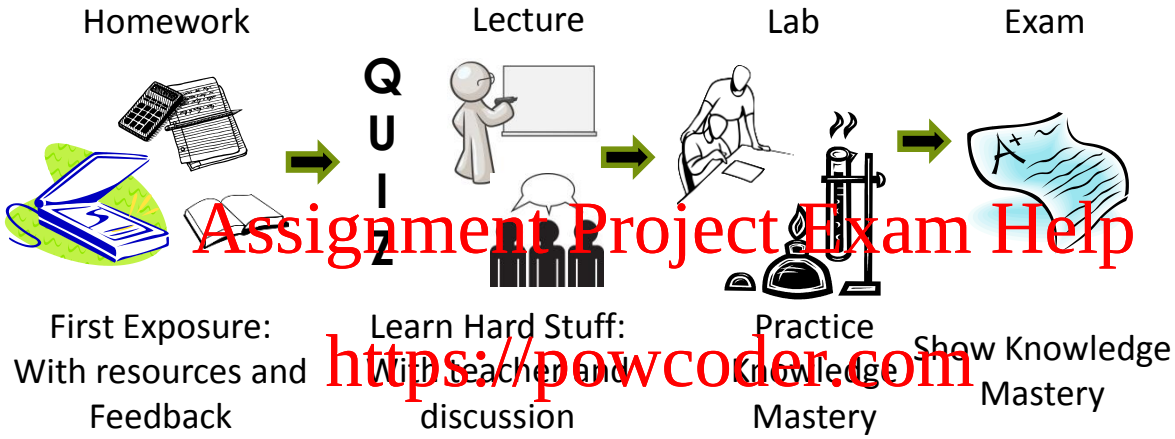
Peer Instruction and Quizzes

- 10% of your total grade will be based on your participation in peer instruction activities. This will include:
  - Quizzes during lectures to review reading. You will receive 1 point for every correct answer you provide.
  - Problem Solving during lectures to break down course material. You will receive 1 point for every answer (**correct or incorrect**) you provide.
- In order to receive full marks for this assessment, you need only to complete 90% of the total possible points.
- http://learn.gla.ac.uk/yacrs/index.php

How does this class work?

- This class is designed a bit differently from what you might normally be used to
  - "Lecture" will be focused around YOU
  - What YOU understand
- A "clicker" is required for this class
  - It's not optional, using it will be 10% of your grade
  - In this class, your "clicker" will be the YACRS system
- So, lecture will be different
  - Ever thought about why we have "lecture"?

Peer Instruction-Based Design



- Greater opportunity for feedback!
- Research on how people learn:
  - Everyone constructs their own understanding
    - We can't dump understanding into your brain
  - To learn YOU must actively work with a problem and construct your own understanding of it

Peer Instruction

- Are you prepared? (quick quiz at beginning of class, using YACRS)
- Pose carefully designed question
  - Solo vote: Think for yourself and select answer
  - Discuss: Analyze problem in teams (but vote independently)
    - Practice analyzing, talking about challenging concepts
    - Reach consensus
    - If you have questions, raise your hand
  - Group vote: Everyone in group votes
  - Class wide discussion:
    - Led by YOU (students) – tell us what you talked about in discussion that everyone should know!

Q1.1: Animal preference

- Go to YACRS now:
  - https//chatsession.dcs.gla.ac.uk
- Join Session 1123

Q1.1: Animal preference

I am a:

- A: Cat Person
- B: Dog Person

- C: I don't like cats or dogs

Q1.1: Animal preference (solution)

I am a:

- A: Cat Person
- **B: Dog Person** ← *I definitely prefer dogs!*
- C: I don't like cats nor dogs

PI 1.1 Software aspects [group vote]

- Go to TACRs now:
  - https//classresponse.gla.ac.uk

- Join Session 1124

PI 1.1: Software aspects [solo vote]

Consider on you own, which aspect of quality is most important in developing a software system for handling medical records in a hospital

- A. Easy of use
- B. Modularity
- C. Modifiability
- D. Efficient
- E. Correct
- F. Maintainability
- G. Understandability
- H. Reusable
- I. Portable

PI 1.2: Software aspects [group vote]

Consider in groups of 3-4 next to you, which aspect of quality is most important in developing a software system for handling medical records in a hospital

- A. Easy of use
- B. Modularity
- C. Modifiability
- D. Efficient
- E. Correct
- F. Maintainability
- G. Understandability
- H. Reusable
- I. Portable

PI 1.1-2: Team Challenge (solution)

- What did you discuss? Anything missing?

Assignment 1

- Modelling and Design using UML (10% of your total grade)

- Full details available on Moodle 19/1 2018

- Due 16/2 2018 at 4:30 on Moodle

Assignment 2

- Software testing (incl design for testing) (10% of your total grade)

- Full details available on Moodle 16/2 2018

- Due 16/3 2018 at 4:30 PM on Moodle

Academic Honesty

- The School has clear policies for academic honesty, if you've forgotten this, it is available on Moodle

- This course depends on discussion and creativity, so collaboration is encouraged, but assignments must be completed individually

- It is your responsibility to be familiar with the policy, but if you are unsure email me

Marking

- All marking schemes will be provided when assignments are announced

- No remarking is allowed except in cases of administrative error.

How to Pass this Class

- Do the readings before coming to the lectures (and perhaps watch the classic one-way lectures on videos)

- Attend the lecture sessions (and be active)

- Attend the labs
- Work Hard
- Ask questions

You are very unlikely to pass this course by reading the slides alone !

Questions?

Comments?

Concerns?

Break (~10 min)

Software Engineering cont.

A Generic Software Project

- Requirements Gathering & Specification
- Design & Modelling
- Construction/implementation (i.e. programming)
- Quality Assurance (i.e. Testing)
- Deployment
- Maintenance & Support

- Project Management (planning, resource allocation, development process, risk estimation/mitigation, cost estimation etc)

Common Activities During Software Projects

- Requirements Specification
  - Domain Analysis
  - Defining the Problem
  - Requirements Gathering
  - Requirements Analysis
  - Requirements Specification

- Covered in Software Project Management course (?)

Common Activities During Software Projects

- Design
  - How should the software be divided into subsystems?
  - How should each subsystem be organised?
  - How should users interact with the system?
  - How should the underlying data be organised and stored?

Common Activities During Software Projects

- Modeling
  - Use case modeling
  - Structural modeling
  - Dynamic and behavioural modeling

Common Activities During Software Projects

- Programming
  - What are the best ways to implement high level designs?

  - Project management: How should teams be organized and managed to successfully create the software ?

Common Activities During Software Projects

- Quality Assurance
  - Design (& program) for testability
  - Reviews and Inspections
  - Testing

Common Activities During Software Projects

- Deployment, Maintenance & Support
  - Release management
  - User documentation
  - Customer support
  - Updates

Common Activities During Software Projects

- Management
  - Resource allocation
  - Cost estimation (based on input from REs)
  - Development models
  - Risk estimation
  - Customer relations
  - … etc

Requirements

PI 1.3-4: Review requirements

- Go to YACRS now:
  - https//chatsession.eng.ac.uk
- Join Session 1125

PI 1.3: Stakeholders [solo vote]

All software engineering projects will have stakeholders. The different stakeholders of a software engineering project will:

- A – all participate in the development of the software
- B – have different priorities and needs for the software
- C – be interested in the development of the software
- D – carry stakes to destroy the software

PI 1.3: Stakeholders (solution)

Stakeholders often have different and sometimes competing needs. For example, consider how the following stakeholders may have different requirements and priorities for a software system:

- Users
- Customers
- Developers
- Managers

PI 1.4 – Requirements [group vote]

The requirements stage is used to understand the customers needs. The requirements specification must primarily describe:

- A: The cost of the system
- B: What the system will do
- C: How the system will be implemented
- D: Who will develop the system

PI 1.4 Requirements (solution)

- The requirements specification must describe what the system will do. Other constraints should only be included if it is unavoidable.

- i.e. **B: What the system will do**

Requirement Specification

- It is a statement describing either
  - 1) an aspect of what the proposed system must do,
  - or 2) a constraint on the system's development.
  - In either case it must contribute in some way towards adequately solving the ***customer's problem***;
  - the set of requirements as a whole represents a negotiated agreement among the stakeholders.

- A collection of requirements is a *requirements document*.

Types of requirements

- Functional requirements
  - Describe *what* the system should do

- Quality requirements
  - *Constraints* on the design to meet specified levels of quality

- Platform requirements
  - *Constraints* on the environment and technology of the system
- Process requirements
  - *Constraints* on the project plan and development methods

Use cases (& scenarios)

- A *use case* is a typical sequence of actions that a user performs in order to complete a given task
  - The objective of use case analysis is to model the system from the point of view of
    … how users interact with the system
    … when trying to achieve their objectives.
    It is one of the key activities in requirements analysis
  - A *use case model* consists of
    - a set of use cases
    - an optional description or diagram indicating how they are related

Use case description

- A. **Name/id**: Give a short, descriptive name to the use case.
- B. **Actors**: List the actors who can perform this use case.
- C. **Goals**: Explain what the actor or actors are trying to achieve.
- D. **Preconditions**: State of the system before the use case.
- E. **Summary**: Give a short informal description.
- F. **Related** use cases.
- G. **Steps**: Describe each step using a 2-column format.
- H. **Postconditions**: State of the system in following completion.

Use case diagrams (modelling)

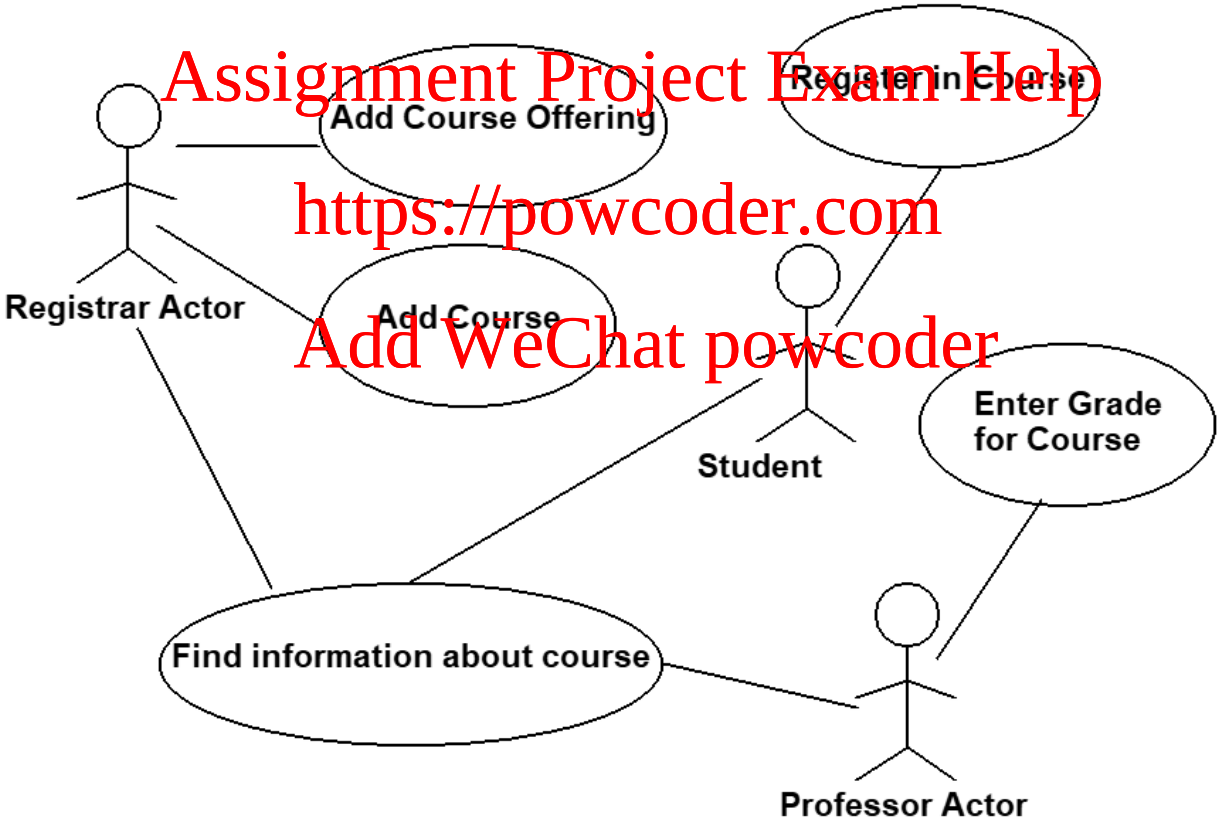


Image: Courtesy of OOSE 2ed

Use case diagrams (modelling)

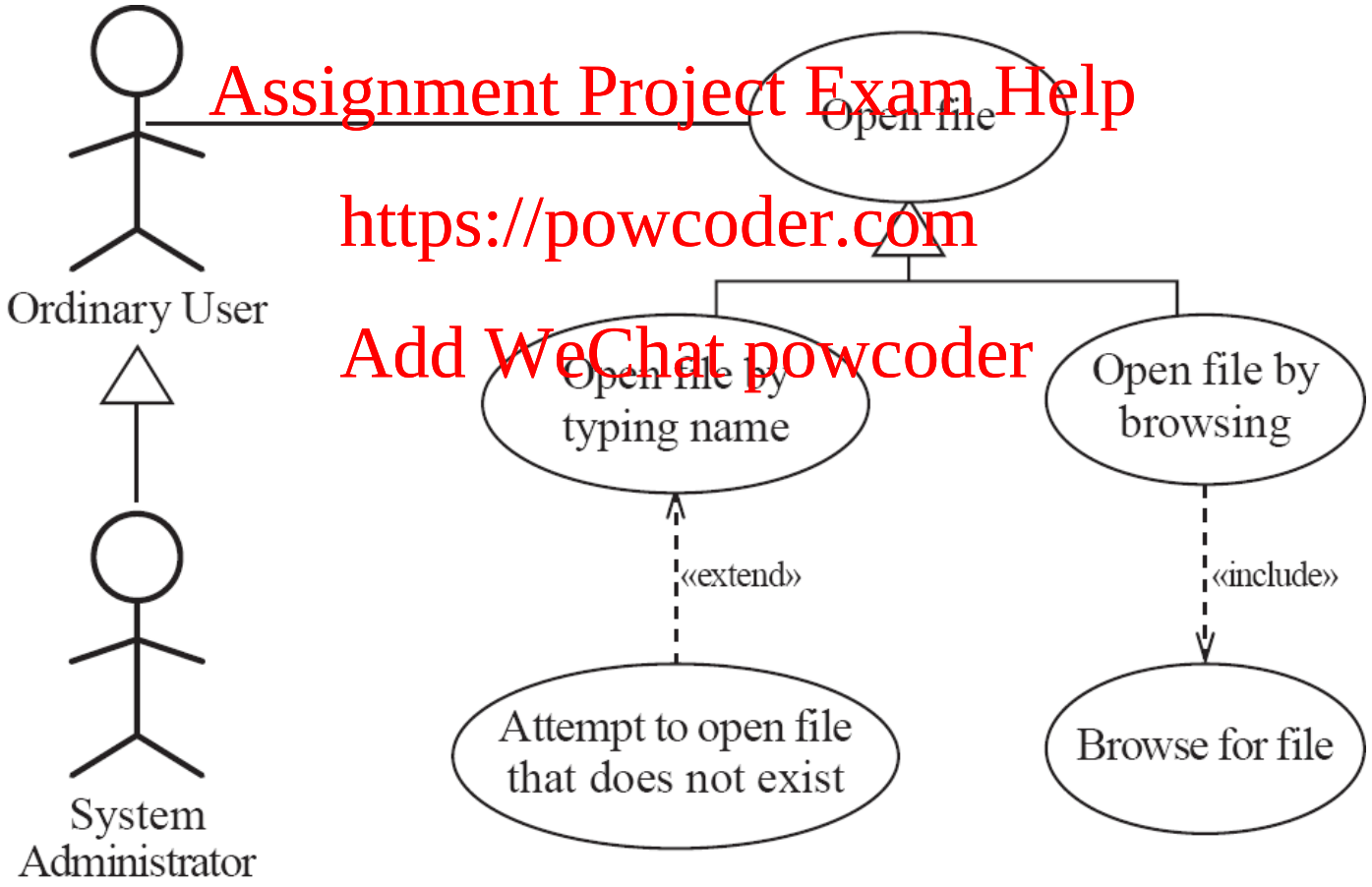


Image: Courtesy of OOSE 2ed

Requirement document

1. Problem
2. Background information
3. Environment and system models
4. Functional Requirements
5. Non-functional requirements

Issues with requirements

- Lack of understanding of the domain or the real problem
  - *Do domain analysis and prototyping*
- Requirements change rapidly
  - *Perform incremental development, build flexibility into the design, do regular reviews*
- Attempting to do too much
  - *Document the problem boundaries at an early stage, carefully estimate the time*
- It may be hard to reconcile conflicting sets of requirements
  - *Brainstorming, group sessions, competing prototypes*
- It is hard to state requirements precisely
  - *Break requirements down into simple sentences and review them carefully, look for potential ambiguity, make early prototypes*

SE (M) 2017-2018

- The focus of this course is to outline principles for how to take a requirements document and come up with a "good" design which when once implemented can be verified to fulfil the requirements (incl. functional, non-functional)

<LONG BREAK>
See you at 14:00 in SAS 208

Bjørn Sand Jensen
bjorn.jensen@glasgow.ac.uk

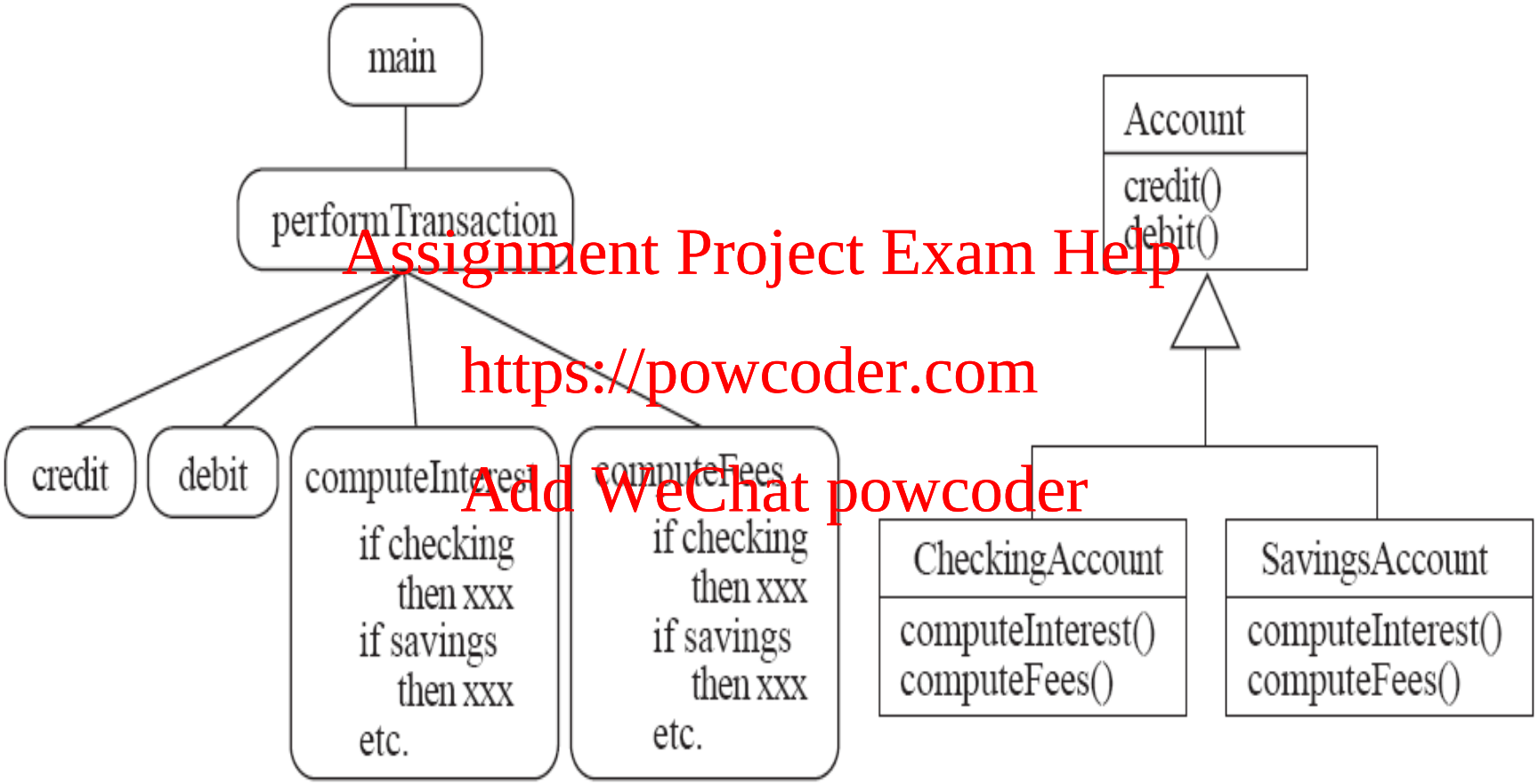
Lecture 1.2 – OO development

(Slides adopted from SE(M) 2016 by Dr Julie Williamson and slides from the OOSE book repository)

OO

- Procedural paradigm:
  - Software is organized around the notion of *procedures*
  - *Procedural abstraction*
    - Works as long as the data is simple
  - *Adding data abstractions*
    - Groups together the pieces of data that describe some entity
    - Helps reduce the system's complexity.
      - Such as *Records* and *structures*

- Object oriented paradigm:
  - Organizing procedural abstractions in the context of data abstractions



OO

- An approach to the solution of problems in which all computations are performed in the context of objects.

  - The objects are instances of classes, which:
    - are data abstractions
    - contain procedural abstractions that operate on the objects
  - A running program can be seen as a collection of objects collaborating to perform a given task

Core OO concepts

- Core OO principles:
  - Encapsulation
  - Inheritance (composition, delegation)
  - Polymorphism
- All allows abstraction…but how ?

SE is not entirely about programming but…

- We assume you are familiar with the following programming/OO concepts (in Java):
  - Variables and assignments (incl. `static` and `final`, and `static final`)
  - Class (incl. `static`, `final` and `static final`)
  - … and objects (instances of classes)
  - `Method` (incl. `private`, `public`, (`protected`), `static`, `final` and `static final`)
  - Attributes
  - Inheritance (`extends`), (simple associations)
  - Basic datatypes (Boolean, Integer, String, arrays,…)
- We will need a few more concepts to do "good" design:
  - Nested classes
  - Abstract classes
  - Interfaces
- We will not rely on Java Generics (albeit useful) and we will not worry (to much) about concurrency

Classes and objects

- Object
  - A chunk of structured data in a running software system

  - Has *properties*

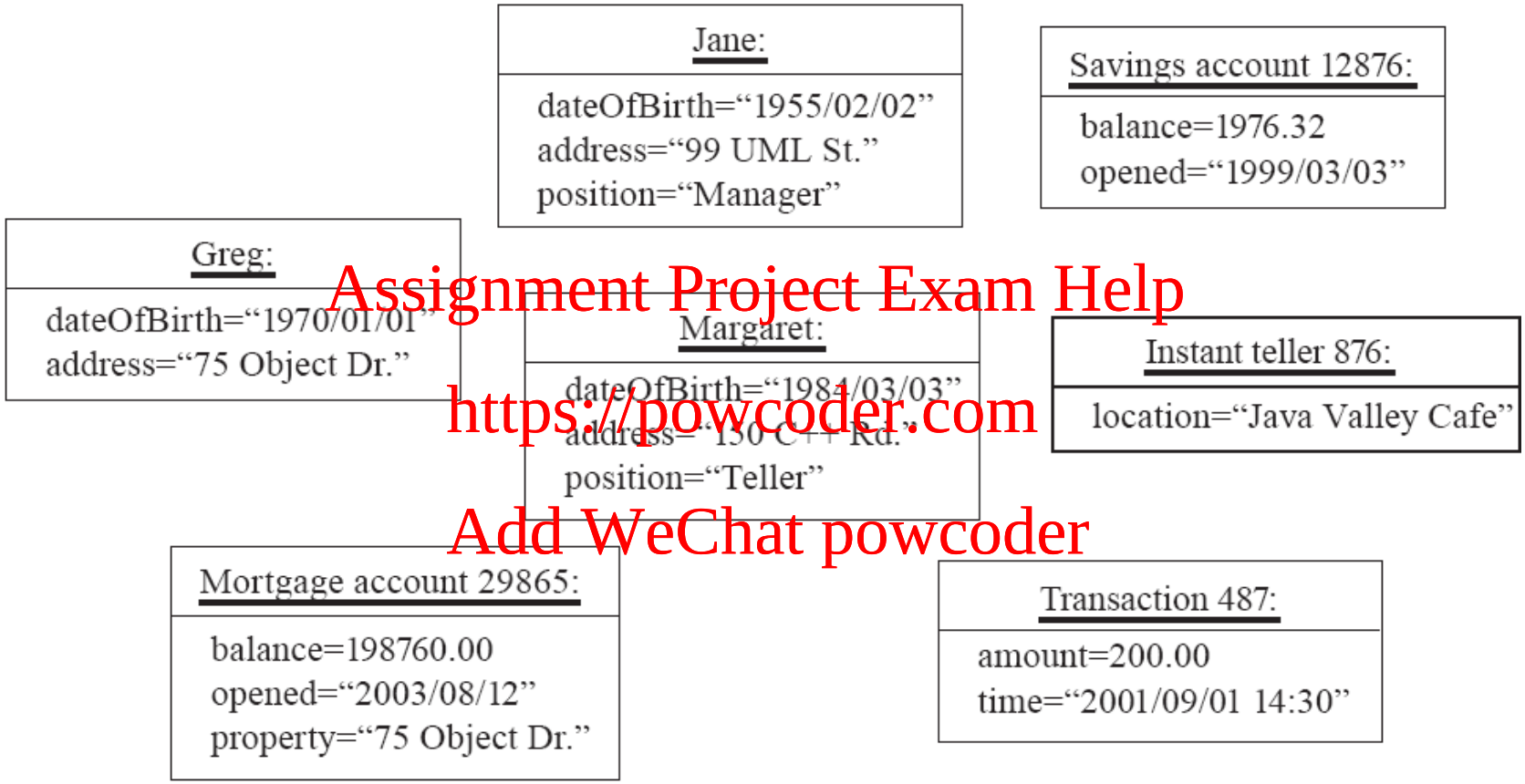
    - Represent its state
  - Has *behaviour*
    - How it acts and reacts
    - May simulate the behaviour of an object in the real world

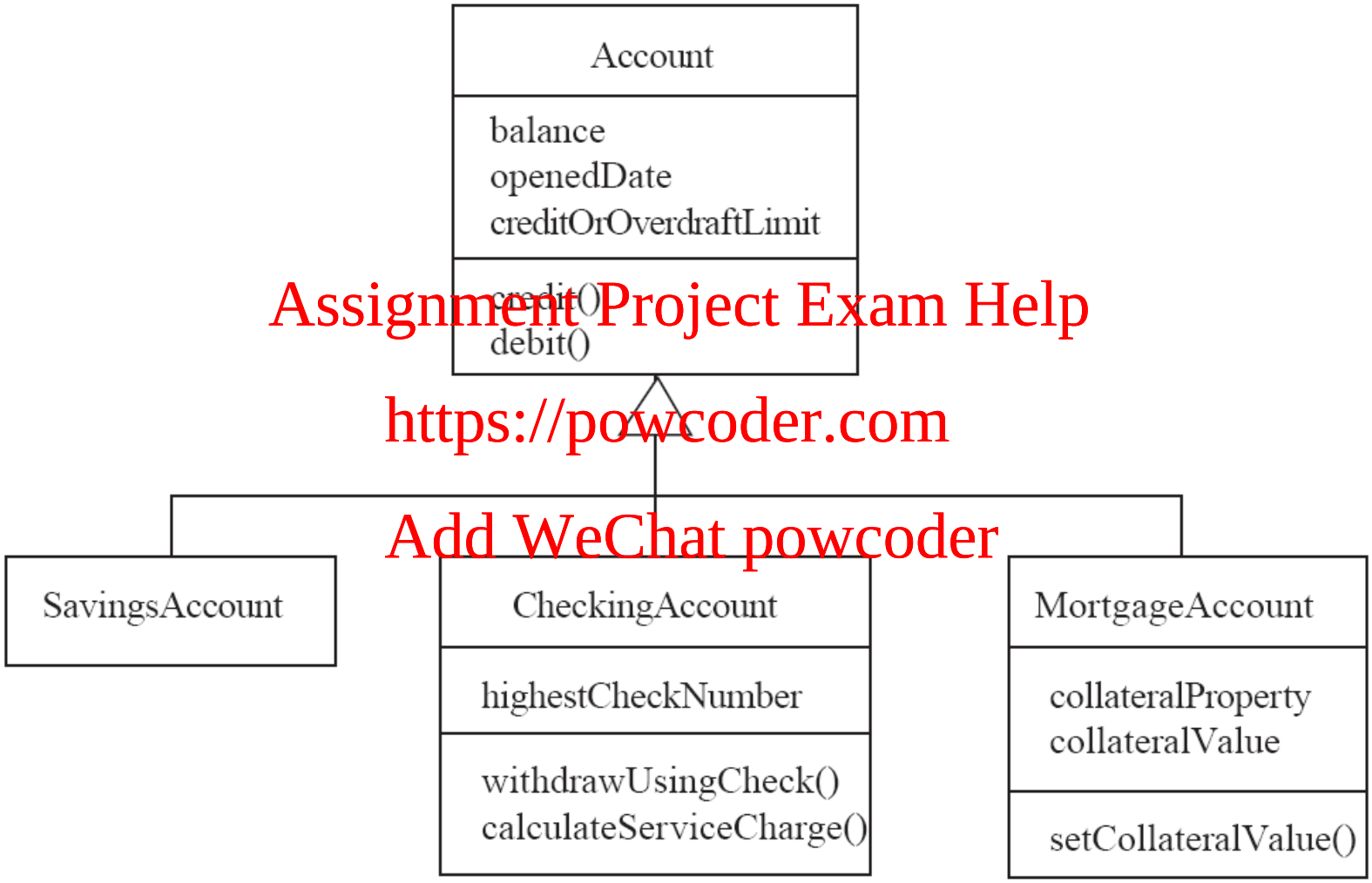
Classes and objects

- A class:
  - A unit of abstraction in an object oriented (OO) program

  - Represents similar objects
    - Its *instances*

  - A kind of software module
    - Describes its instances' structure (properties)
    - Contains *methods* to implement their behaviour





PI 1.5: Objects and Java

- Go to YACRS now:
  - https//classresponse.cs.gla.ac.uk
- Join Session 1126

PI 1.5 - Objects and Java

In this code, bob is a:

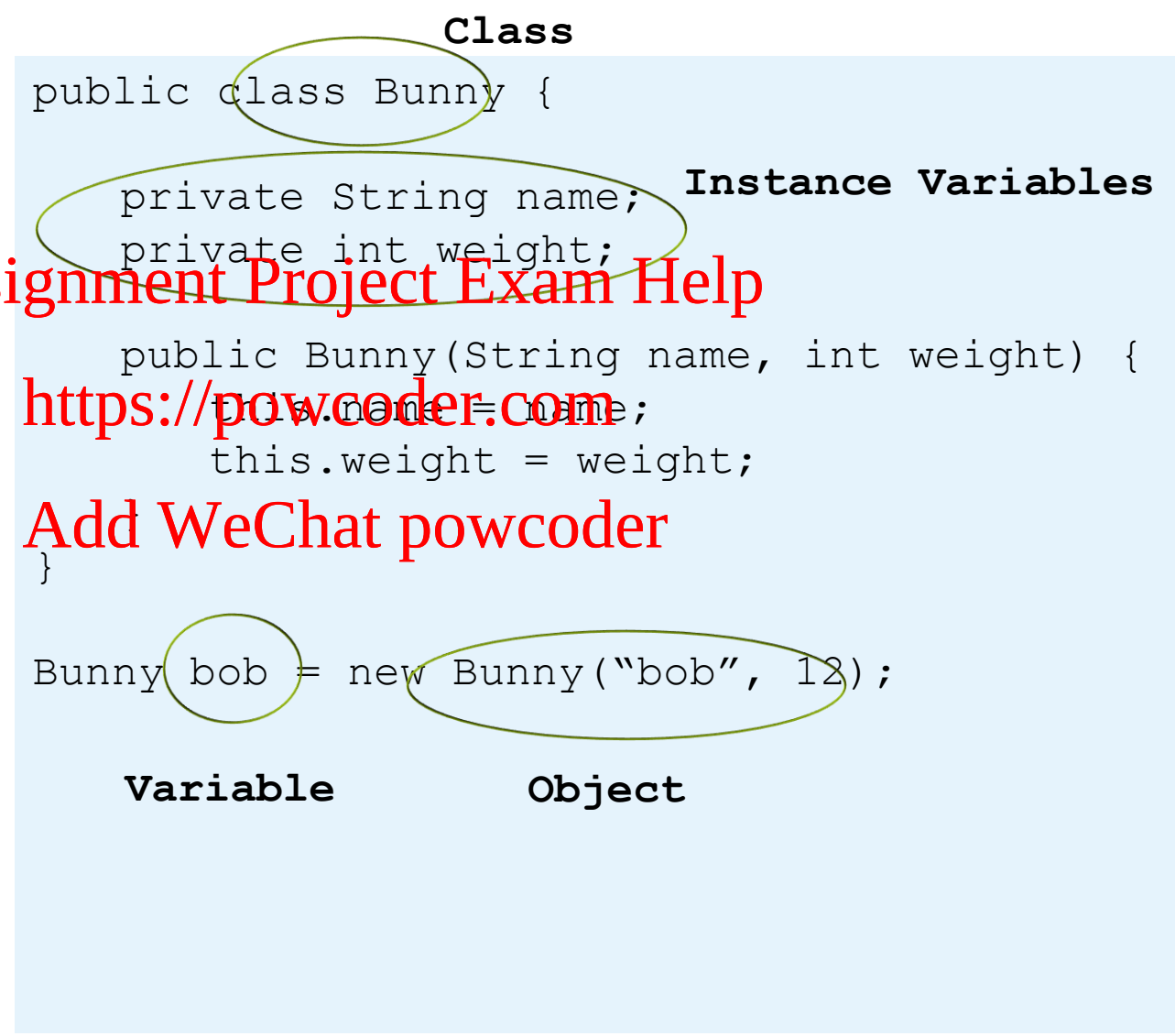
- A: Class
- B: Object
- C: Variable
- D: Abstraction

```
public class Bunny {

    private String name;
    private int weight;

    public Bunny(String name, int weight) {
        this.name = name;
        this.weight = weight;
    }
}

Bunny bob = new Bunny("bob", 12);
```

PI 1.5 - Objects and Java (sol.)

In this code, bob is a:

- A: Class
- B: Object
- **C: Variable**
- D: Abstraction



Methods

- Method
  - A procedural abstraction used to implement the behaviour of a class

  - Several different classes can have methods with the same name
    - They implement the same abstract operation in ways suitable to each class
    - E.g. calculating area in a rectangle is done differently from in a circle

Polymorphism

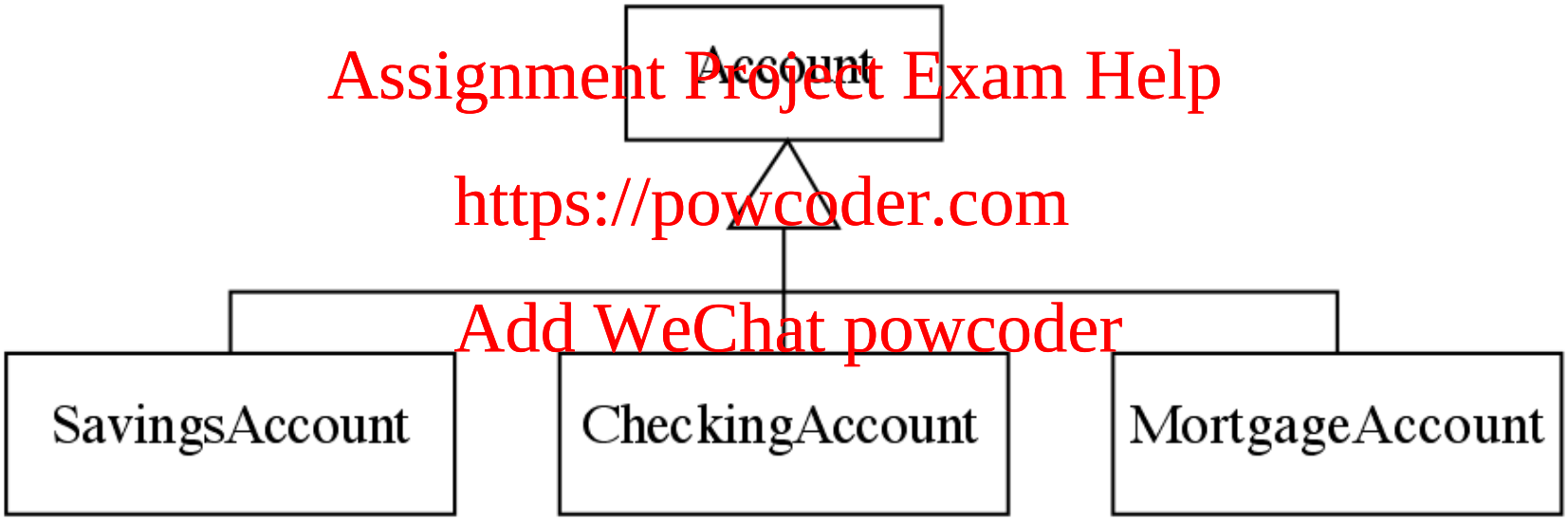
- A property of object oriented software by which an *abstract operation may be performed in different ways* in different classes.

  - Requires that there be *multiple methods of the same name*

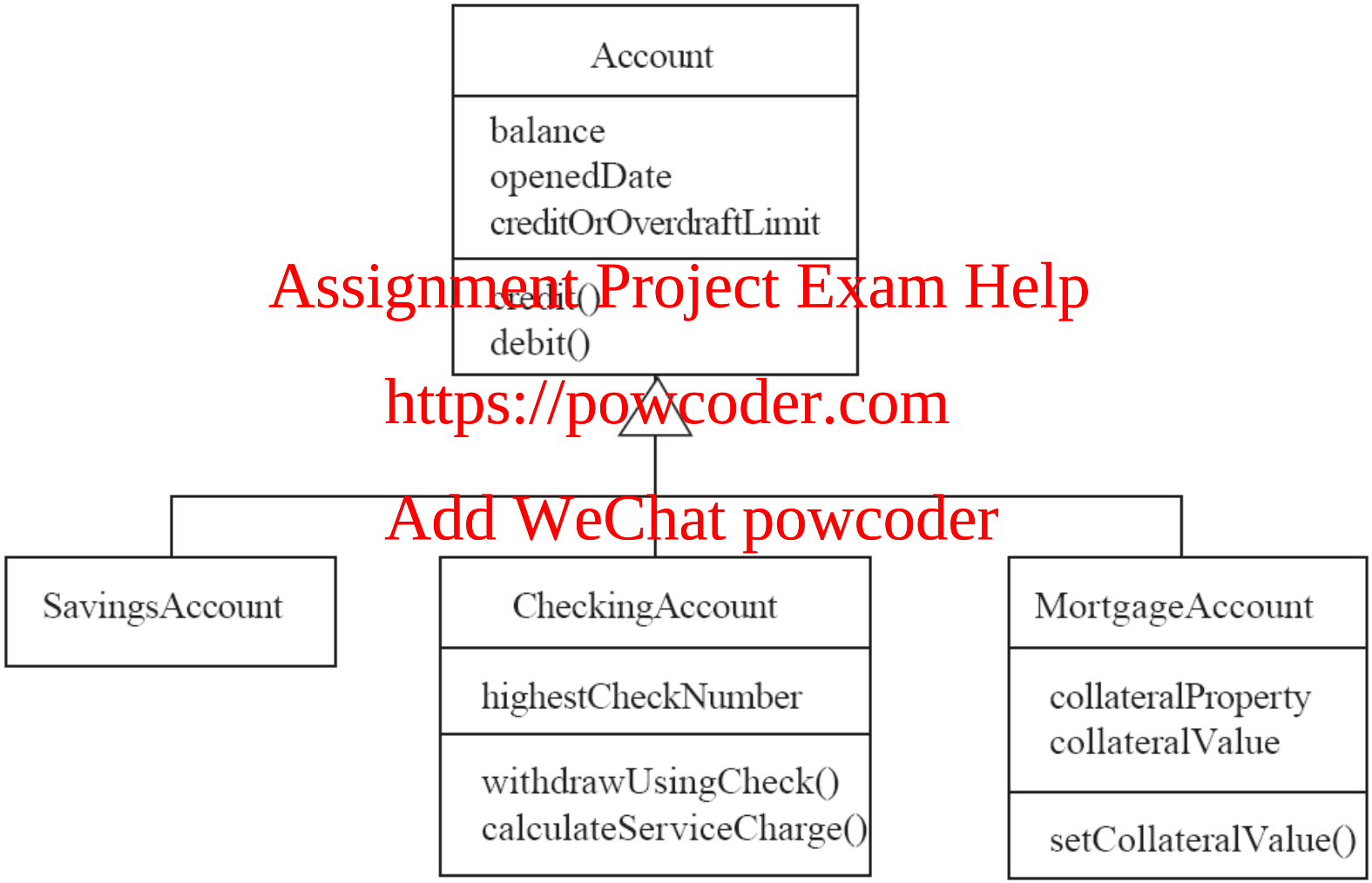
  - The choice of which one to execute depends on the object that is in a variable
  - Reduces the need for programmers to code many `if-else` or `switch` statements

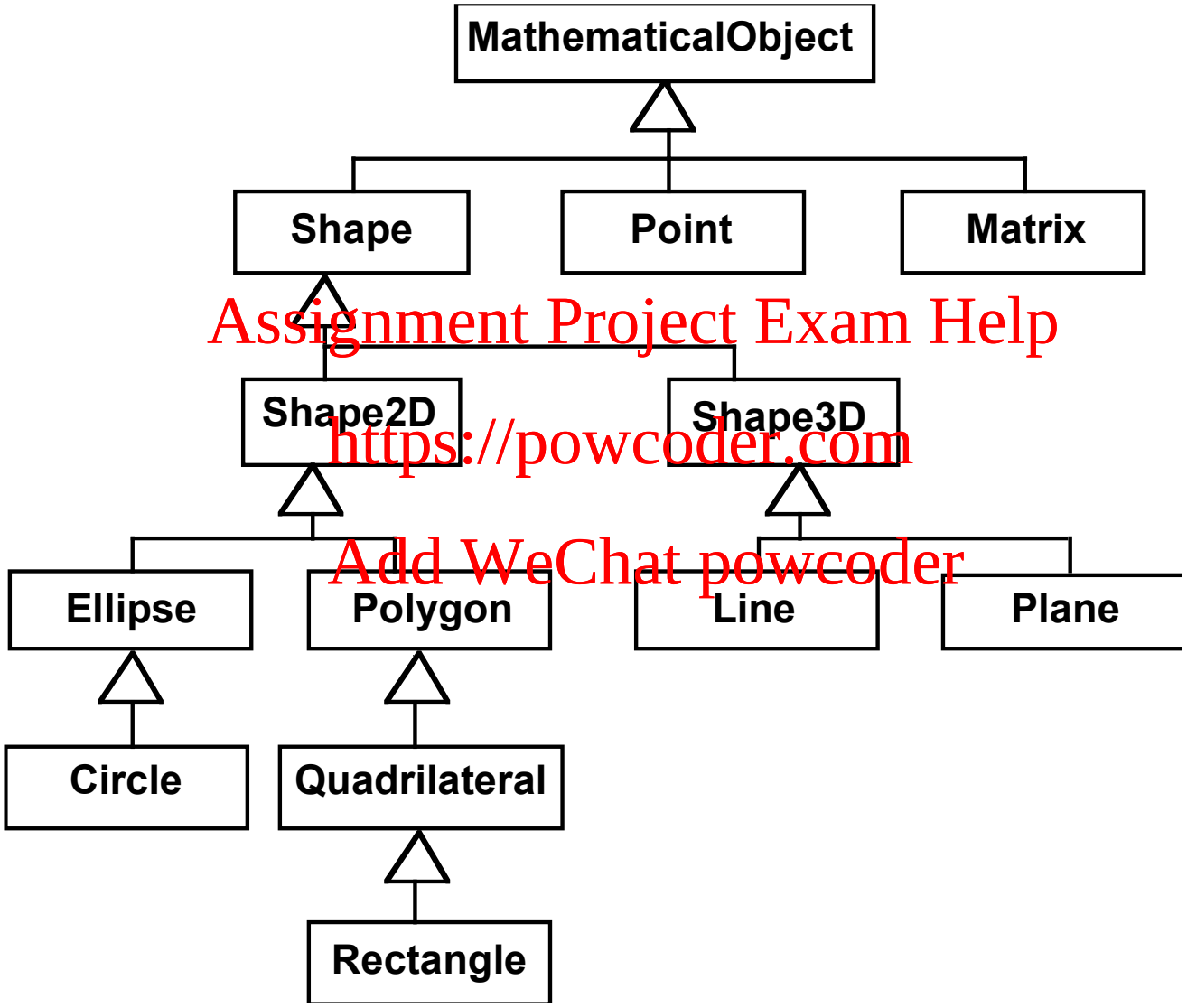
Inheritance Hierarchies

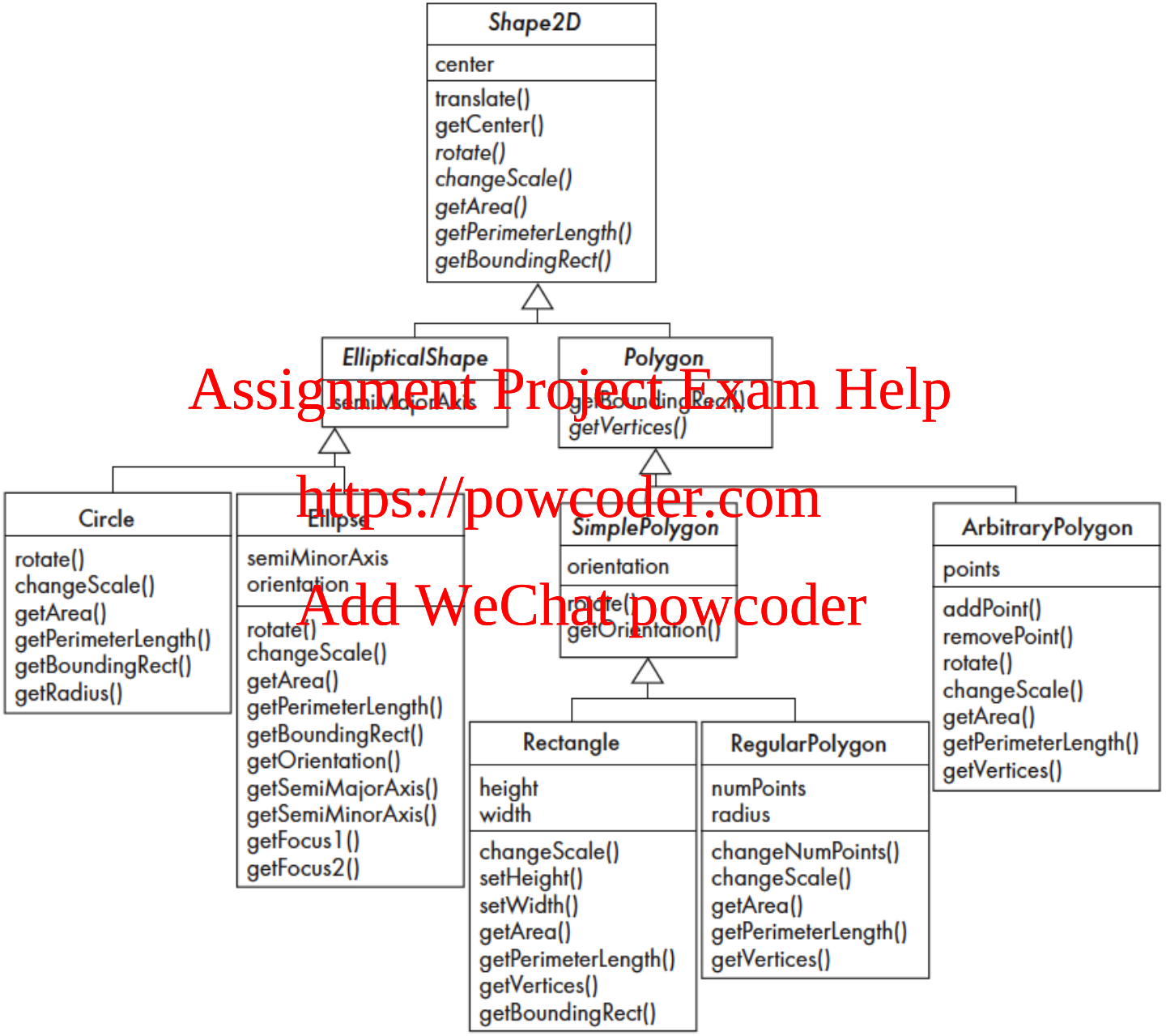
- Superclasses
  - Contain features common to a set of subclasses
- Inheritance hierarchies
  - Show the relationships among superclasses and subclasses
  - A triangle shows a *generalization*
- Inheritance
  - The *implicit* possession by all subclasses of features defined in its superclasses



- Inheritance
  - The *implicit* possession by all subclasses of features defined in its superclasses







Abstract Classes (and methods)

- Use when you don't want the class type to be able to be instantiated

- Declare methods as `abstract` if you want to enforce the inheriting type

- Abstract classes can give inherited functionality with non-abstract methods

Example: Fruit

```
public abstract class Fruit {

        public abstract boolean isEdible();

        public void takeABite() {

                if (this.isEdible()) {

                        System.out.println("Tasty Fruit.");

                }

                else {

                        System.out.println("You are about to be sick.");

                }

        }

}
```

Interface

- Like abstract classes, but cannot have executable statements
  - Define a set of operations that make sense in several possibly unrelated classes (e.g. Comparable)
- A class can implement any number of interfaces
  - It must have concrete methods for the operations
- You can declare the type of a variable to be an interface
  - This is just like declaring the type to be an abstract class
- Important interfaces in Java's library include
  - Runnable, Collection, Iterator, Comparable, Cloneable

Overriding

- A method would be inherited, but a subclass contains a new version instead
  - For restriction
    - E.g. `scale(x,y)` would not work in `Circle`

  - For extension
    - E.g. `SavingsAccount` might charge an extra fee following every debit
  - For optimization
    - E.g. The `getPerimeterLength` method in `Circle` is much simpler than the one in `Ellipse`

PI 1.7: OO

- Go to YACRS now:
  - https//chatsession.dcs.gla.ac.uk
- Join Session 1127

PI 1.7: OO

This code could be refactored to better utilise object orientation. How would you change this code to make it more efficient?

A: Create an Abstract Class UIElement with an abstract method draw

B: Create a Class UIElement with a method draw

C: Create an Interface drawable with a method Draw

D: Create an Interface with methods for draw, getX, and getY

```
public class Button {
    int x;
    int y;
    String text;
    public int getX() {
        return this.x;
    }
    public int getY() {
        return this.y;
    }
    public void draw() {
        //  Do some swing
    }
}
public class Radio {
    int x;
    int y;
    String text;
    public int getX() {
        return this.x;
    }
    public int getY() {
        return this.y;
    }
    public void draw() {
        //  Do some swing
    }
}
```

PI 1.7: OO (solution)

This code could be refactored to better utilise object orientation. How would you change this code to make it more efficient?

A: **Create an Abstract Class UIElement with an abstract method draw**

← This class follow the "is-a" rule, a radio button **is a** UIElement

B: Create a Class UIElement with a method draw

← In this case, we don't want a generic UIElement to be able to be instantiated

C: **Create an Interface drawable with a method Draw**

← These elements could all be clustered as "drawable" things

D: Create an Interface with methods for draw, getX, and getY

← It is context dependent if this is usefull. It implies that all drawable has x and y coordinates which is not a very usefull abstraction. Could be interpreted as implementing the actual methods which is not allowed on interfaces (clearly wrong in that case).

Access

- Applies to methods and variables
  - `public`
    - Any class can access
  - `protected`
    - Only code in the package or subclasses can access
  - (blank)
    - Only code in the package can access
  - `private`
    - Only code written in the class can access
    - Inheritance still occurs!
- See e.g. https://docs.oracle.com/javase/tutorial/java/landl/subclasses.html

Basic documentation

- Comment extensively
  - Comment whatever is non-obvious
  - Do not comment the obvious
  - Comments should be 25-50% of the code

- Organize class elements consistently
  - Variables, constructors, public methods then private methods

- Be consistent regarding layout of code

Main idea: Coupling and Cohesion

- Cohesion
  - How related things in a given class are (within a class)

- Coupling
  - How dependent classes are on each other (between classes)

- Design principle: **increase cohesion** and **decrease coupling**

Wrap up (Week 1)

- Reading: Chapters 1 and 2 (should be familiar, but material will be examinable).

- Requirements is needed before we can start designing our system (they may change). Review Chapter 4.

- OO approach allows abstraction
- Encapsulation, Inheritance/composition and polymorphism are core concepts of OO-based design and development.

Preparation for Next Week

- We'll be covering UML and modelling
- Read OOSE Chapter 5 Modeling Classes
- Assessed Quiz (starting ***precisely*** at 10:05) will be on **UML syntax**

Questions?

Comments?

Concerns?