Review Lectures

The Final Exam Paper

- Duration: 2 hours and 30 minutes
- Reading: 15 minutes
- Total marks: 65
- Hurdle: 32.5

The Structure

<table>
<thead>
<tr>
<th>Sections</th>
<th>Questions</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Choice</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Short Answer</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>Long Answer</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>Total</td>
<td>34</td>
<td>65</td>
</tr>
</tbody>
</table>

Topics

- Software Engineering (=5% of 65)

Topics (Cont’d)

- Java and OOP (=65% of 65)
  - Concepts & Definitions
  - Code fragments
  - Writing a small program

Topics (Cont’d)

- UML and OOD (=30% of 65)
  - Concepts & Definitions
  - Understanding Diagrams
  - OOD Principles
  - Design Patterns
  - Design using UML
Software Engineering - Introduction

- Software Engineering is an engineering discipline which is concerned with all aspects of software production from the early stages of system requirements through to maintaining the system after it has gone into use.

Software Process

- Software Process defines the way to produce software. It includes
  - Software life-cycle model
  - Tools to use
  - Individuals building software
  - Software life-cycle model defines how different phases of the life cycle are managed.

Phases of Software Life-cycle

- Requirements
- Specification (Analysis)
- Design
- Implementation
- Integration
- Maintenance
- Retirement

Life-Cycle Models

- Build-and-fix model
- Waterfall model
- Rapid prototyping model
- Incremental model
- Extreme programming
- Synchronize-and-stabilize model
- Spiral model
- Object-oriented life-cycle models
- Comparison of life-cycle models

Abstract Data Type (ADT)

- A structure that contains both data and the actions to be performed on that data.

- Class is an implementation of an Abstract Data Type.

Object Oriented Design Concepts
**Class**

- Class is a set of attributes and operations that are performed on the attributes.

<table>
<thead>
<tr>
<th>Account</th>
<th>Student</th>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountName</td>
<td>name</td>
<td>centre</td>
</tr>
<tr>
<td>accountBalance</td>
<td>age</td>
<td>radius</td>
</tr>
<tr>
<td>withdraw()</td>
<td>studentID</td>
<td>area()</td>
</tr>
<tr>
<td>deposit()</td>
<td>getName()</td>
<td>circumference()</td>
</tr>
<tr>
<td>determineBalance()</td>
<td>getId()</td>
<td></td>
</tr>
</tbody>
</table>

**Objects**

- An Object Oriented system is a collection of interacting Objects.

- Object is an instance of a class.

**Classes/Objects**

- John and Jill are objects of class Student

- circleA and circleB are objects of class Circle

**Object Oriented Paradigm: Features**

- Encapsulation
- Data Abstraction
- Inheritance
- Polymorphism
- Persistence
- Delegation

**Java Review**

```java
// HelloWorld.java: Hello World program

class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World");
    }
}
```
Program Processing

- **Compilation**
  
  # javac HelloWorld.java
  
  results in HelloWorld.class

- **Execution**
  
  # java HelloWorld
  
  Hello World

Basic Data Types

- **Types**
  
  - boolean
  - either true or false
  - char
  - 16 bit Unicode 1.1
  - byte
  - 8-bit integer (signed)
  - short
  - 16-bit integer (signed)
  - int
  - 32-bit integer (signed)
  - long
  - 64-bit integer (singed)
  - float
  - 32-bit floating point (IEEE 754-1985)
  - double
  - 64-bit floating point (IEEE 754-1985)
  - String (class for manipulating strings)
  - Java uses Unicode to represent characters internally

Control Flow

- **Control Flow Statements in JAVA**
  
  - while loop
  - for loop
  - do-while loop
  - if-else statement
  - switch statement
  - JAVA does not support a goto statement

Classes

- **A class is a collection of fields (data) and methods (procedure or function) that operate on that data.**
- **The basic syntax for a class definition:**

  ```java
  public class Circle {
    // my circle class
  }
  ```

- **Bare bone class – no fields, no methods**

Constructors

- **Constructor is a method that gets invoked at object creation time.**
- **Constructors have the same name as the class.**
- **Constructors cannot return values.**
- **Constructors are normally used for initializing objects.**
- **A class can have more than one constructor – with different input arguments.**
**Defining a Constructor**

- Like any other method

```java
public class ClassName {
    // Data Fields...
    // Constructor
    public ClassName() {
        // Method Body Statements initialising Data Fields
    }
    // Methods to manipulate data fields
}
```

- Invoking:
  - There is NO explicit invocation statement needed: When the object creation statement is executed, the constructor method will be executed automatically.

**Method Overloading**

- Constructors all have the same name?
- In Java, methods are distinguished by:
  - name
  - number of arguments
  - type of
  - position of arguments
- Not method overriding (coming up), method overloading:

**Polymorphism**

- Allows a single method or operator associated with different meaning depending on the type of data passed to it. It can be realised through:
  - Method Overloading (Supported in C++, but not in Java)
  - Operator Overloading
  - Defining the same method with different argument types (method overloading) - polymorphism.
  - The method body can have different logic depending on the date type of arguments.

**Scenario**

- A Program needs to find a maximum of two numbers or Strings. Write a separate function for each operation.
  - In C:
    - int max_int(int a, int b)
    - int max_string(char *s1, char *s2)
    - max_int (10, 5) or max_string ("melbourne", "sydney")
  - In Java:
    - int max(int a, int b)
    - int max(String s1, String s2)
    - max(10, 5) or max("melbourne", "sydney")
- Which is better? Readability and intuitive wise?

**Data Hiding and Encapsulation**

- Java provides control over the visibility of variables and methods, encapsulation, safely sealing data within the capsule of the class
- Prevents programmers from relying on details of class implementation, so you can update without worry
- Keeps code elegant and clean (easier to maintain)

**Visibility**

- Circle
  - Construction time message
  - Center (x,y)
  - Radius
  - Area
  - Message
Parameter passing
- Method parameters which are objects are passed by reference.
- Copy of the reference to the object is passed into method, original value unchanged.

Delegation
- Ability for a class to delegate its responsibilities to another class.
- A way of making an object invoking services of other objects through containership.

Inheritance
- Ability to define a class as a subclass of another class.
- Subclass inherits properties from the parent class.

Subclassing
- Subclasses created by the keyword `extends`.
- Each `GraphicCircle` object is also a `Circle`.

Abstract Classes
- An `Abstract` class is a conceptual class.
- An Abstract class cannot be instantiated – objects cannot be created.
- Abstract classes provides a common root for a group of classes, nicely tied together in a package.
Abstract Classes

```java
public Circle extends Shape {
    protected double r;
    protected static final double PI = 3.1415926535;
    public Circle () { r = 1.0; } // Constructor
    public double area() { return PI * r * r; }
    // Other methods...
}

public Rectangle extends Shape {
    protected double w, h;
    public Rectangle () { w = 0.0; h = 0.0; } // Constructor
    public double area() { return w * h; }
    // Other methods...
}
```

Interfaces

- **Interface** is a conceptual entity similar to an Abstract class.
- Can contain only constants (final variables) and abstract method (no implementation) - Different from Abstract classes.
- Use when a number of classes share a common interface.
- Each class should implement the interface.

Interface - Example

```
<<Interface>>
Speaker
  speak()
Politician
  speak()
Priest
  speak()
Lecturer
  speak()
```

Interfaces: An informal way of realising multiple inheritance

- An interface is basically a kind of class—it contains methods and variables, but they have to be only abstract classes and final fields/variables.
- Therefore, it is the responsibility of the class that implements an interface to supply the code for methods.
- A class can implement any number of interfaces, but cannot extend more than one class at a time.
- Therefore, interfaces are considered as an informal way of realising multiple inheritance in Java.

Interfaces Definition

- **Syntax (appears like abstract class):**
  ```java
  interface InterfaceName {
      // Constant/Final Variable Declaration
      // Methods Declaration – only method body
  }
  ```
- **Example:**
  ```java
  interface Speaker {
      public void speak();
  }
  ```
Error Handling

- Any program can find itself in unusual circumstances – Error Conditions.
- A “good” program should be able to handle these conditions gracefully.
- Java provides a mechanism to handle these error condition – exceptions

Exceptions in Java

- A method can signal an error condition by throwing an exception – throws
- The calling method can transfer control to a exception handler by catching an exception - try, catch
- Clean up can be done by - finally

Common Java Exceptions

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- FileNotFoundException
- IOException – general I/O failure
- NullPointerException – referencing a null object
- OutOfMemoryException
- SecurityException – when applet tries to perform an action not allowed by the browser’s security setting.
- StackOverflowException
- StringIndexOutOfBoundsException

Syntax of Exception Handling Code

```java
...
...
try {
    // statements
} catch ( Exception-Type e ) {
    // statements to process exception
} ...
```

Streams

- Java Uses the concept of Streams to represent the ordered sequence of data, a common characteristic shared by all I/O devices.
- Streams presents a uniform, easy to use, object oriented interface between the program and I/O devices.
- A stream in Java is a path along which data flows (like a river or pipe along which water flows).
I/O and Data Movement

- The flow of data into a program (input) may come from different devices such as keyboard, mouse, memory, disk, network, or another program.
- The flow of data out of a program (output) may go to the screen, printer, memory, disk, network, another program.
- Both input and output share a certain common property such as unidirectional movement of data - a sequence of bytes and characters and support to the sequential access to the data.

Stream Types

- The concepts of sending data from one stream to another (like a pipe feeding into another pipe) has made streams powerful tool for file processing.
- Connecting streams can also act as filters.
- Streams are classified into two basic types:
  - Input Stream
  - Output Stream

Java Stream Classes

- Input/Output related classes are defined in java.io package.
- Input/Output in Java is defined in terms of streams.
- A stream is a sequence of data, of no particular length.
- Java classes can be categorised into two groups based on the data type one which they operate:
  - Byte streams
  - Character Streams

Classification of Java Stream Classes

Graphical User Interface (GUI) Applications

- Abstract Windowing Toolkit (AWT)
- Events Handling
- Applets

AWT - Abstract Windowing Toolkit

- Single Windowing Interface on Multiple Platforms
- Supports functions common to all window systems
- Uses Underlying Native Window system
- AWT provides
  - GUI widgets
  - Event Handling
  - Containers for widgets
  - Layout managers
  - Graphic operations
Building Graphical User Interfaces

- import java.awt.*;
- Assemble the GUI
  - use GUI components,
    - basic components (e.g., Button, TextField)
    - containers (Frame, Panel)
  - set the positioning of the components
    - use Layout Managers
- Attach events

A sample GUI program

```java
import java.awt.*;
public class MyGui {
    public static void main(String args[])
    {
        Frame f = new Frame("My Frame");
        Button b = new Button("OK");
        TextField tf = new TextField("Programming in Java", 20);
        f.setLayout(new FlowLayout());
        f.add(b);
        f.add(tf);
        f.setSize(300, 300);
        f.setVisible(true);
    }
}
```

Output

Sockets and Java Socket Classes

- A socket is an endpoint of a two-way communication link between two programs running on the network.
- A socket is bound to a port number so that the TCP layer can identify the application that data destined to be sent.
- Java's .net package provides two classes:
  - Socket – for implementing a client
  - ServerSocket – for implementing a server

Java Sockets

A Multithreaded Program

It can be host_name like “mandroo.cs.mu.oz.au”
An example

class MyThread extends Thread {  // the thread
    public void run() {  
        System.out.println("this thread is running ... ");
    }
}  // end class MyThread

class ThreadEx1 {  // a program that utilizes the thread
    public static void main(String [] args) {
        MyThread t = new MyThread();  
        // due to extending the Thread class (above)
        // I can call start(), and this will call
        // run(). start() is a method in class Thread.
        t.start();
    }  // end main()
}  // and class ThreadEx1

Life Cycle of Thread

Software Development Process and Unified Modeling Language (UML)

- A software development process is a set of phases that are followed to bring a product or a system from conception to delivery.
- In the Unified Process, there are four of these phases:
  - Inception (Analysis phase): identify the system we are going to develop, including what it contains and its business case.
  - UML: use-case diagrams
  - Elaboration (Design phase): perform detailed design and identify the foundation of system from "use case diagram", which eventually lead to classes.
  - UML: classes, objects, class diagrams, sequence diagram, collaboration diagrams etc.
  - Construction (Implementation phase): write software using Java/C++
    - the actual building of the product from the design of the system
  - Transition (Rolling out phase): Deliver the system product to the users. Includes maintenance, upgrades, and so on until phasing out.

Unified Modeling Language

Unified Modeling Language diagrams:
- Structural diagrams: Class Diagram, Object Diagram, Component Diagram, Deployment Diagram
- Behavioral diagrams: Use case Diagram, Sequence Diagram, Collaboration Diagram, Statechart Diagram, Activity Diagram

Use Case Diagrams

- Use Case diagrams show the various activities the users can perform on the system.
- System is something that performs a function.
- They model the dynamic aspects of the system.
- Provides a user's perspective of the system.
Use Case Diagrams

- **ACTORS**: roles users can play in interacting with the system.
  - An actor is used to represent something that users our system.
- **USE CASES**: each describes a possible kind of interaction between an actor and the system.
  - Uses cases are actions that a user takes on a system
- **RELATIONSHIPS** between these entities (Actors and Use Cases).
  - Relationships are simply illustrated with a line connecting actors to use cases.

Use Case Diagrams - Actors

- An *actor* is a user of the system playing a particular role.
- Actor is shown with a stick figure.

Use Case Diagrams – Use Cases

- Use case is a particular activity a user can do on the system.
- Is represented by an ellipse.
- Following are two use cases for a library system.

Use Case Diagram – Example1 (Library)

Class Visibility

- public level +
- protected level #
- private level -

```
Circle
- centerX: Int
- centerY: Int = 0
+ draw()
# move(Int X, Int Y)
```
Association

- Association describes a link, a link being a connection among objects between classes.
- Association is shown by a solid line between classes.

Association - Example

- A Person works for a Company.

Generalization (Inheritance)

- Child class is a special case of the parent class

Abstract Methods (Operations)

- Shape
  - draw()

Realization - Interface

- Interface is a set of operation the class carries out

Class Diagram Example

- School
  - has
  - member
  - offeredBy
  - assignedTo

- Department
  - 1

- Student
  - attends

- Course
  - teaches
Sequence Diagram

- Shows how objects communicate with each other over time.
- The sequence diagram consists of **OBJECTS**, **MESSAGES** represented as solid-line arrows, and **TIME** represented as a vertical progression.

Types of Messages

- Synchronous (flow interrupt until the message has completed).
- Asynchronous (don’t wait for response)
- Flat – no distinction between syns/async
- Return – control flow has returned to the caller.

Sequence Diagram – Compilation

- **Compiler**
- **Linker**
- **FileSystem**

Sequence Diagram – Time & Messages

- Messages are used to illustrate communication between different active objects of a sequence diagram.

Sequence Diagram – Enroll Student for subject successfully

Collaboration Diagram – Enroll Student in Subject Scenario

- **U:URSDatabase**
- **procCmd(cmd)**
- **parseCommand(cmd)** (transient)
- **AsgSubCmd(cmd)**
- **execute**
- **addSubject(sub)**