ALGONQUIN COLLEGE

CST8132 OBJECT ORIENTED PROGRAMMING

Testing
Javadoc
Midterm Preparation

Professor : Dr. Anu Thomas
Email: thomasa@algonquincollege.com
Office: T314

Today's Topics

- Labs
  - Lab 3 – Inheritance, abstract class, interface
- Hybrids
- Midterm
  - Week 7 – in class
  - MCQs
  - Worth 10% of term marks

Midterm Preparation

- On June 24 at 10:00 AM, during lecture
- You have 1.5 hours (90 minutes) to complete it
- There will not be a lecture, as soon as you are done you may leave
- There will be around 50 MCQ questions…
  - Multiple choice, multi-select, ordering, fill in the blanks etc.
- You need to join zoom by 9:50 AM.
- Webcam should be on for the full duration… I can see your activities in zoom and in Brightspace
- I will check IP address from zoom & Brightspace, and if they are not matching, you will get a 0

Midterm Preparation

- CAL Students : must send their CAL letters by Friday June 18, 11:59 PM.

What should you learn?

- Encapsulation
- Polymorphism
- Abstraction
- Inheritance
- Abstract class
- Overriding/Overloading
- Interface
- ArrayList
- Multi-dimensional arrays

- this keyword
- super keyword
- final keyword
- Static keyword
- Access specifiers
- Constructors
  - default,
  - no-arg
  - parameterized
- Relationships
  - Association
  - Aggregation
  - Composition
  - Cardinality

- Variables
  - Local
  - instance
- Packages
- Testing
- Javadoc

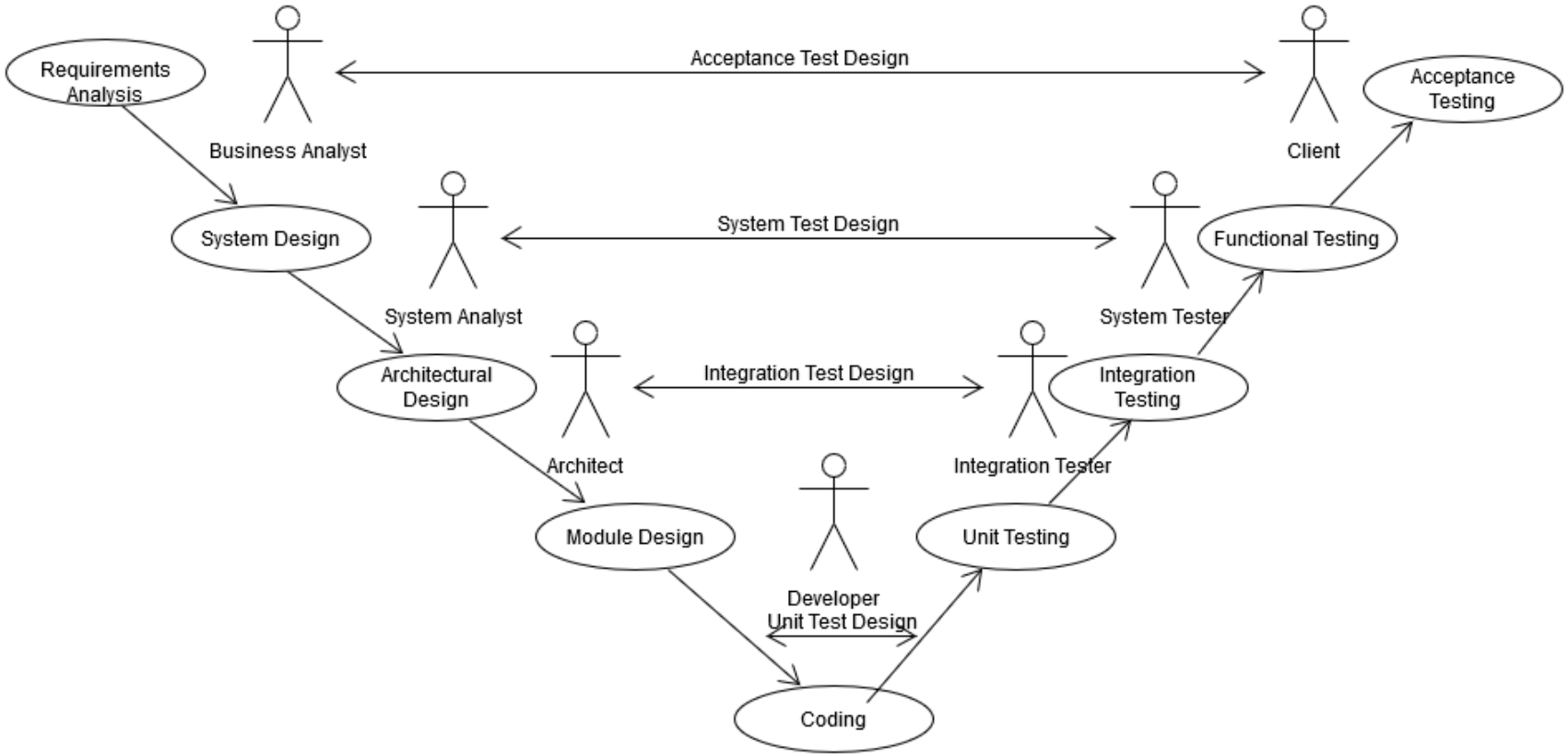
Recap

1. Inheritance
   - Super-class, sub-class
   - Super keyword – super(), super.method()
2. Polymorphism
   - Taking multiple forms
   - Static polymorphism – compile time (ex. Method overloading)
   - Dynamic polymorphism – runtime (ex. Method overriding)
3. Abstract class
   - Providing abstract of methods…. Will be overridden in child classes
   - Can have non-abstract methods – they can have their definitions in the abstract class
4. Interface
   1. Only abstract methods and public static final variables
   2. Abstract keyword not required – by default, methods are abstract

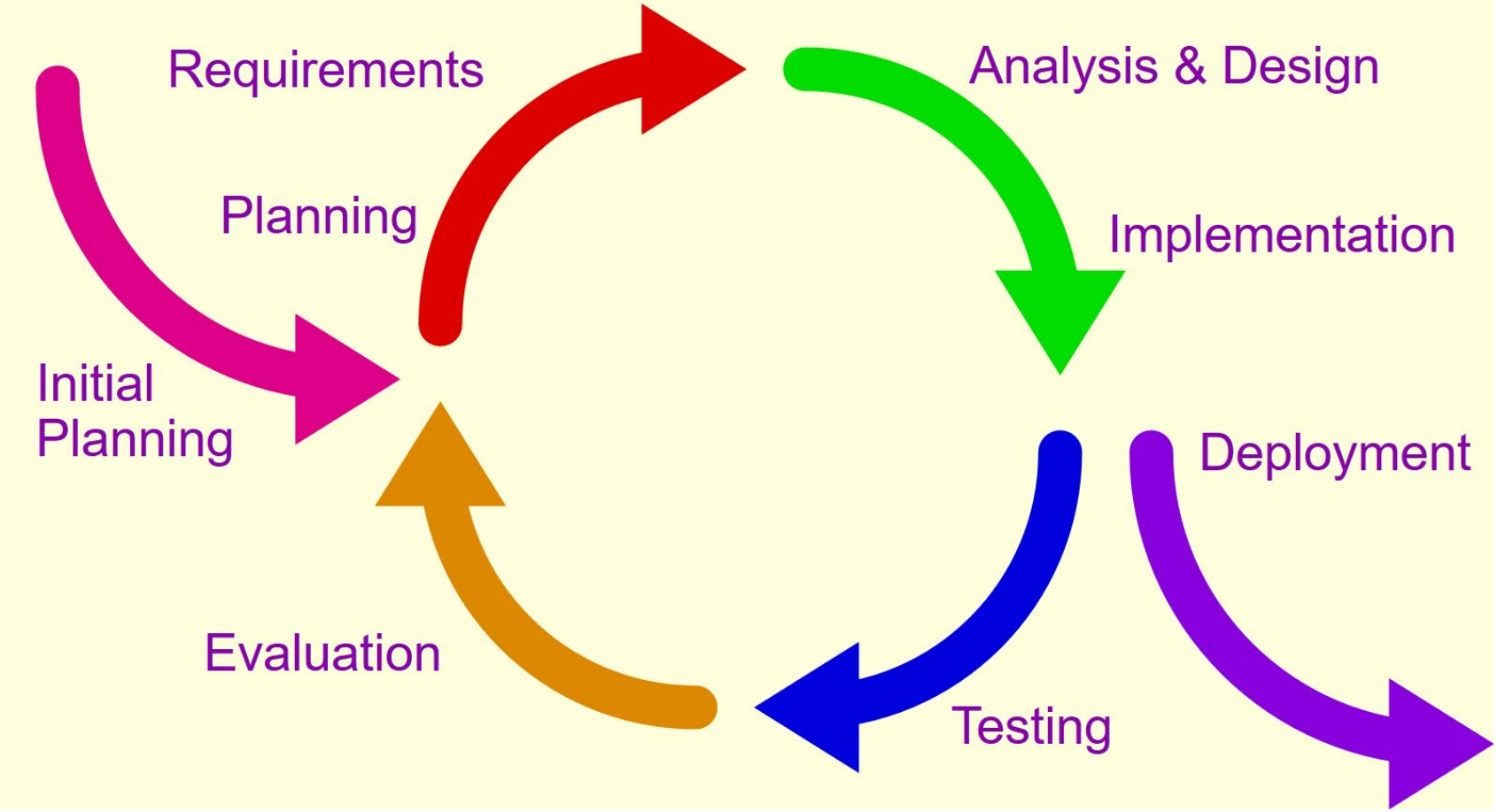
Test Plans

- A test plan is a comprehensive strategy for ensuring that the system does not contain errors:
  - Unit testing:
    - testing the individual units of our programs (classes, methods)
  - Integration testing:
    - putting the units together, and testing their use of each other's interfaces etc
  - Functional testing:
    - tests whether the overall system matches the behaviour defined in the system requirements
  - Acceptance testing
    - Evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

V model (Waterfall)



Iterative Development Model (Agile)



https://en.wikipedia.org/wiki/Iterative_and_incremental_development

Boundary Conditions

- In developing good tests for your code, you will need to identify all the boundary conditions
- You want to test with data that probes the boundary conditions
- What are the boundaries in an array?

Int [] num = new int[100];

- Array indices: -1, 0 , 1
- Array indices: 99, 100
- Boundaries on int values? Integer.MAX_VALUE, Integer.MIN_VALUE, int values change sign: -1, 0, 1

Sample Test Plan

CST8132 – Object-Oriented Programming

Test Plan < – write project name>

Prepared By - <Student Name>

<add more tables if you have more features to test. Add more rows if you have more test cases>

Feature: Menu Processing

Test #	Description	Input	Expected output/result	Actual Output	Status (Pass/Fail)
T1	Invalid selection – negative number	-1	Invalid Option—please try again…	Invalid Option—please try again…	Pass
	Invalid input - string	n	Invalid Option—please try again…	InputMismatch – program crashed	Fail
T7	Valid selection	1			

Feature: <Name of the feature>

Test #	Description	Input	Expected output/result	Actual Output	Status (Pass/Fail)

JUnit

- Junit is a facility for automated unit testing
- It supports test classes that run your methods and “look for the unexpected”
- Your tests will use the static methods of the Assert class to assert the condition you are testing for
  - Example

- See sections 1 – 4 of http://www.vogella.com/tutorials/JUnit/article.html

Summary

- Testing – How you are going to test if the program is working properly
  - Unit – lowest level testing (Class and methods)
  - Integration - test the integration between a set of components
  - Functional/System – test if the system works as a whole
- JUnit – Automated tool to unit test your code

Javadoc

- documentation generator created for the Java language
- intended for other developers, and represents the API of your Java class.
- produces formatted HTML documentation easily for your code.
- can produce automatic documentation from your code, however you can supplement this using special comments throughout your code.

Javadoc

- Javadoc comments must be enclosed by
  - /** */
  - They MUST begin with /**
- Javadoc comments MUST appear on the line(s) immediately preceding the code they describe.
- Wherever a description is required, a proper descriptive sentence or sentence fragment is expected.
- Javadoc is REQUIRED on all assignments going forward.
- Javadoc is not a replacement for comments.

Javadoc

- consist of descriptive text, and tagged descriptors.
- tags begin with an @ symbol, and always use lower-case names.
- Some tags are required, and some are optional. Some may be conditionally repeated.
- conditionally required whether you are documenting classes, properties or methods.
- Class documentation does not replace the required header as described in requirements.

Javadoc

- Java Class Documentation
  - Every class requires the following Javadoc to appear immediately above the class declaration (below any package or import statements).

```
/**
  * The purpose of this class is ...
  * @author  John Doe
  * @version 1.0
  * @since   1.8
 **/
 public class MyClass {
```

Javadoc

- Java Property Documentation
  - Every non-private property (instance or class variable) requires a Javadoc description to appear immediately above the variable declaration.

```
/** The current account balance. **/
protected double balance;

/** The unique account number. **/
protected long accountNum;

/** The client who is the account holder. */
public Client client;
```

Javadoc

- Java Property Documentation
  - Declare every property on a new line.
  - Private properties do not require Javadoc.
  - Do not simply repeat the variable name – this is not sufficient. Javadoc gets the name and data type automatically. Your job is to write qualitative information about the property.
  - The valid range of values is a good thing to include in Javadoc documentation. For example:

```
/** The hour of day, 0 - 23. */
protected int hour;
```

Javadoc

- Java Method Documentation
  - Every non-private method requires a Javadoc description to appear immediately above the method declaration.
  - Constructors and the main method require Javadoc!!!
  - Methods with parameters require the Javadoc to contain one **@param** tag for each parameter.
  - Methods with a return type require the Javadoc to contain one and only one **@return** tag for the return value.
  - When we begin exception handling, methods which throw exceptions should have an **@throws** tag for each exception type thrown.

Javadoc

- Java Method Documentation
  - Example:

```
/** This method attempts to withdraw a
   * specified amount from the account.
   * @param amt
   *        The amount requested to withdraw.
   * @return True if the withdrawal is
   *         successful, or false if the
   *         balance is insufficient for the
   *         amount requested.
 **/
public boolean withdraw ( double amt ) {
```

Javadoc

- Javadoc Requirements
  - [description]
  - @author [author name]
    - If more than one author, each author has their own @author tag. Order from most recent author at the top of the list to the oldest at the end of the list.
  - @version [software version]
    - For your assignments, use a sensible version such as 1.0.
  - @since [jdk version]
    - We are using version 1.8. Could provide full version.
  - @param [name] [description]
    - The name of the parameter must match the Javadoc name.
  - @return [description]

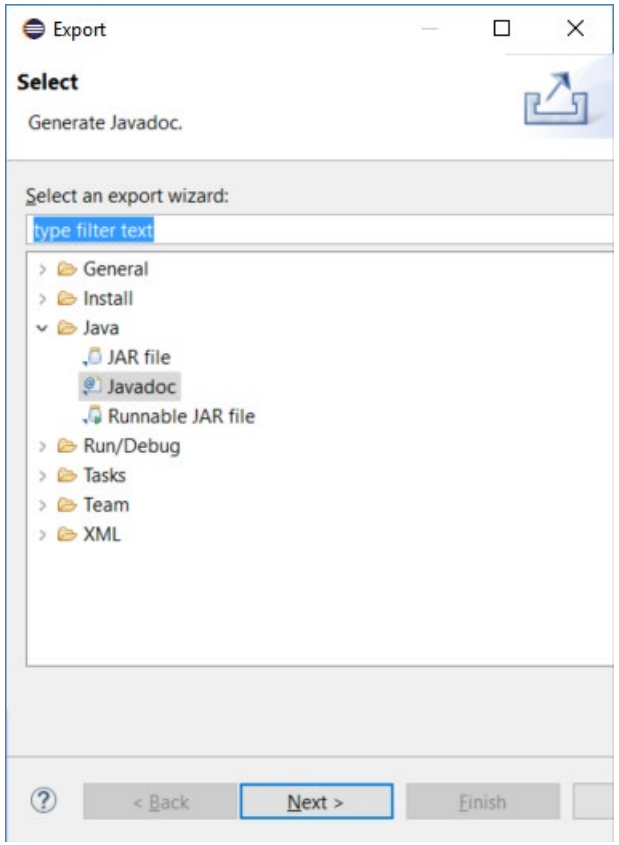
Javadoc

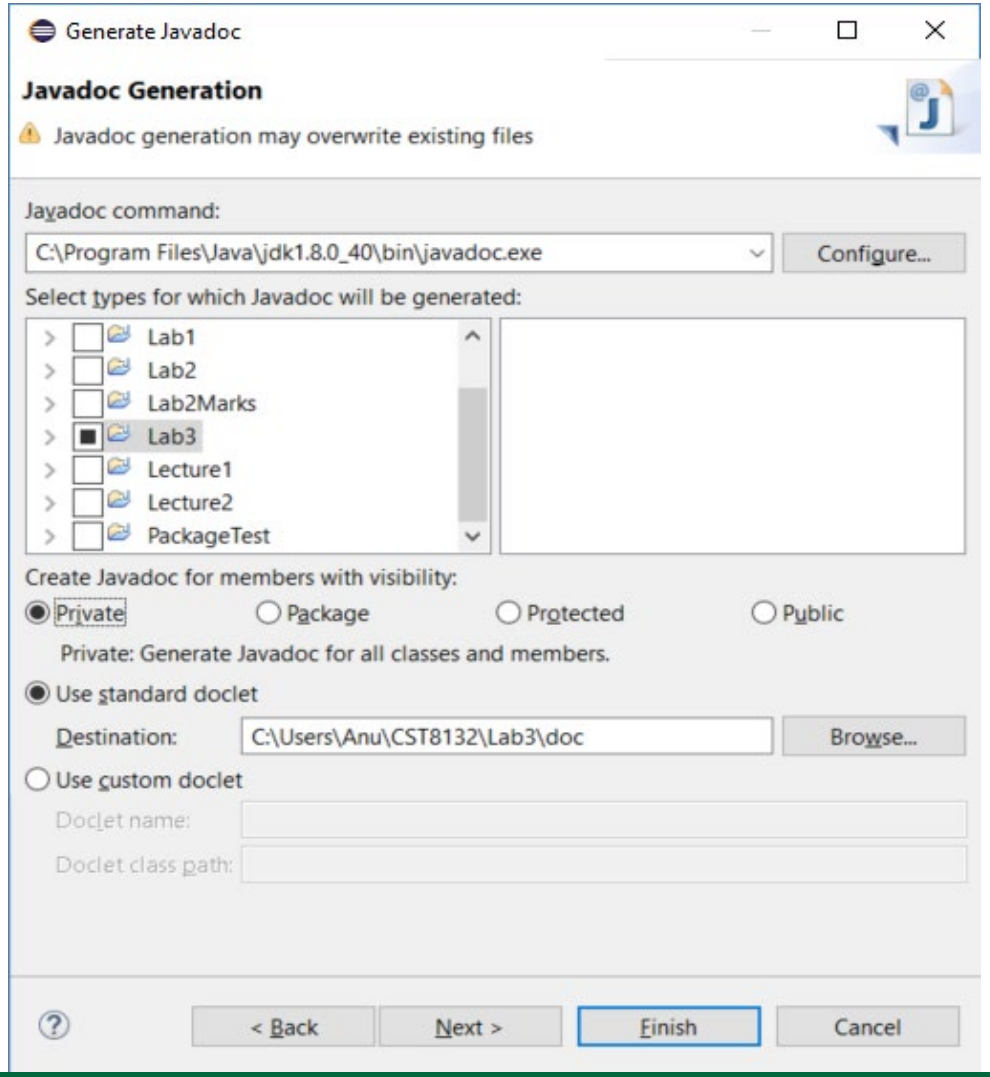
- Javadoc Details
  - Javadoc descriptions may contain HTML mark-up for additional formatting opportunities.
  - Javadoc does NOT replace the need for comments in your code.
    - Code blocks such as control structures still require comments.
  - Javadoc should not appear inside of any methods – only comments belong inside of methods.
- To enable Javadoc validation in Eclipse:
  - Window > Preferences > Java > Compiler > Javadoc

Generating Javadoc

- Javadoc will only be generated based on comments within /** .... */ frame.
- To generate Javadoc in Eclipse:
  - Select the **File → Export → Java→ Javadoc** menu option.
  - Click the **Configure…** button next to the Javadoc Command text box, and **browse** to your installed javadoc.exe file. For example:
    - C:\Program Files\Java\jdk1.8.0_40\bin\javadoc.exe.
  - Ensure that the project and ALL java files within are selected.
  - From **Create Javadoc for members with visibility**, select the **Package** option.
  - Leave the **Destination** with the default value.
  - Click **Finish**.

Generating JavaDoc





What does JavaDoc output look like?

- API Documentation - https://docs.oracle.com/en/java/javase/11/docs/api/index.html#
- String - https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html