UNIVERSITY OF
WESTMINSTER

5COSC022W.2 Client Server Architectures

Tutorial Week 01: Java Objects, Classes, Exceptions, and Logging

In this tutorial, we will review some important concepts in Java that are essential in this module.

Requirements:

- NetBeans IDE

Section 1

1.1 LEARN: CLASS

A class is a blueprint or a template for creating objects. It defines the properties (attributes or fields) and behaviors (methods) that objects of the class will have. Here's a simple example of a class in Java:

```java
// Define a class called "Car"
public class Car {
    // Fields or attributes
    String make;
    String model;
    int year;

    // Constructor to initialize the object
    public Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }

    // Method to display information about the car
    public void displayInfo() {
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }
}
```

In this example, we've defined a class named **Car** with three fields (**make**, **model**, and **year**), a constructor to initialize these fields, and a method **displayInfo()** to print information about the car.

1.2 LEARN: OBJECTS

An object is an instance of a class. It is created based on the blueprint provided by the class. Here's an example of creating objects from the **Car** class:

```java
public class Main {
    public static void main(String[] args) {
        // Create objects of the Car class
        Car car1 = new Car("Toyota", "Camry", 2020);
        Car car2 = new Car("Honda", "Civic", 2021);

        // Access and modify object properties
        car1.displayInfo(); // Display information about the first car

        // Modify the year of the second car
        car2.year = 2022;

        // Display information about the second car after modification
        car2.displayInfo();
    }
}
```

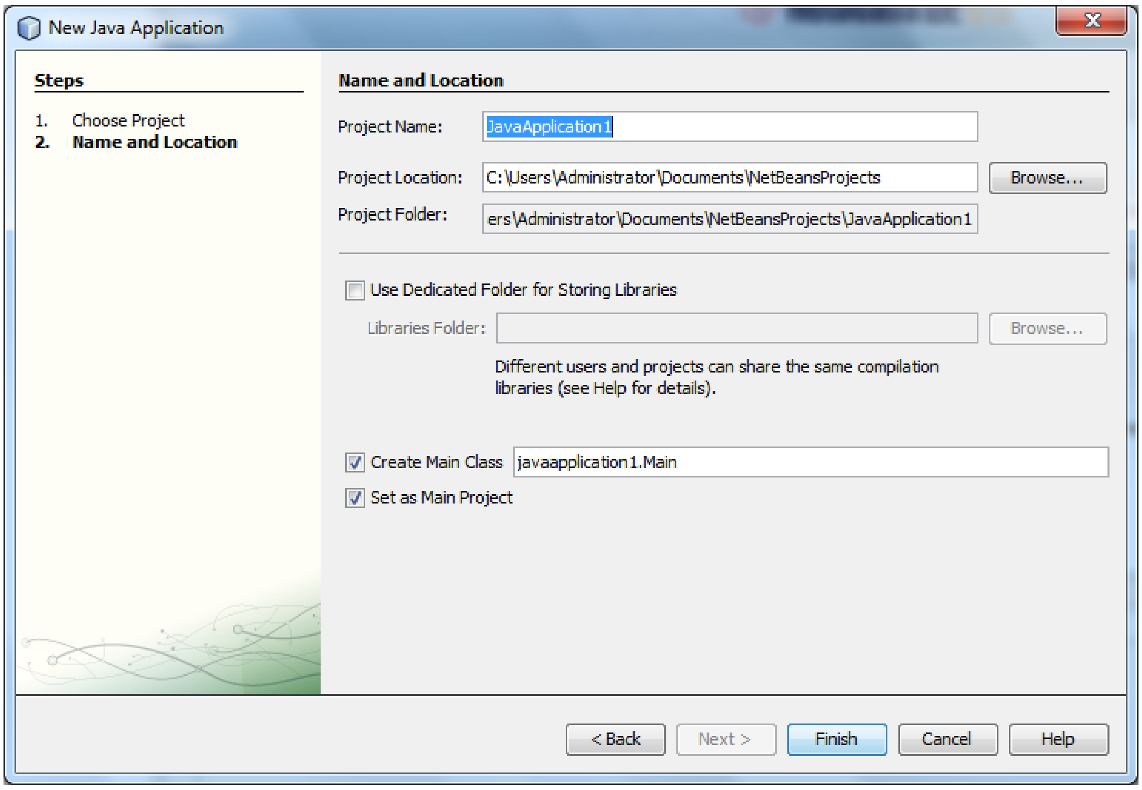
In this example, we create two objects (**car1** and **car2**) of the **Car** class using the **new** keyword. We then use these objects to access and modify the properties of the cars and invoke the **displayInfo()** method to print information.

Please note that the classes provide a blueprint for creating objects, and objects are instances of classes that encapsulate data (fields) and behavior (methods). OOP principles, such as encapsulation, inheritance, and polymorphism, can be applied to create robust and modular code.

To test this example, you may create a new project and name it as Main. Once you implement the codes, you can run the project and share the result with your instructor.

1.3 LEARN: CREATING A PROJECT IN NETBEANS

1. Click on File option in Menu Bar or you can directly click on the New Project option in the Tool Bar or you can press Ctrl + N for starting the New Project in java.
2. A **New Project** window will appear in front of your screen.
3. Select the **Categories** as **JAVA** and in the **Projects** Select **Java Application**.
4. Then Press the Next Button in the window.
5. Enter the name of the Project "**but keep in mind that while writing the name of the project space is not required. Enter the name without leaving the space.**"
6. Finally, click on **Finish** to create the project.


New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: JavaApplication1

Project Location: C:\Users\Administrator\Documents\NetBeansProjects

Project Folder: ers\Administrator\Documents\NetBeansProjects\JavaApplication1

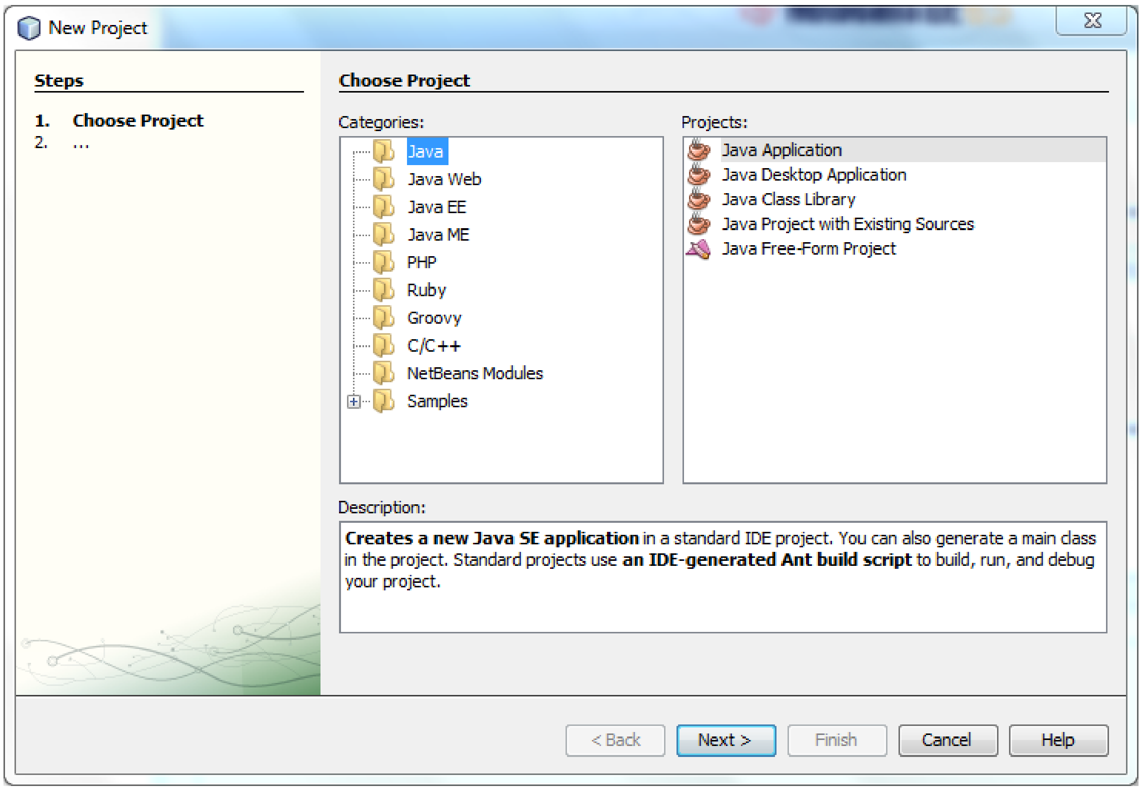
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Main Class javaapplication1.Main

☑ Set as Main Project


New Project

Steps

1. **Choose Project**
2. ...

Choose Project

Categories:

- Java
- Java Web
- Java EE
- Java ME
- PHP
- Ruby
- Groovy
- C/C++
- NetBeans Modules
- Samples

Projects:

- Java Application
- Java Desktop Application
- Java Class Library
- Java Project with Existing Sources
- Java Free-Form Project

Description:

Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

1.4 EXERCISE 1: MODELLING A SIMPLE SENDER/RECEIVER PROGRAM

The purpose of this exercise is to simulate a client-server architecture with simple Java codes. To do this, please create a new Java project called **Tutorial01Message** and create four classes: Message.java, Receiver.java,Sender.java,MessagePassingExample.java

Step 1: Create the **Message** Class

1.1. Open your preferred Java development environment (IDE)

1.2. Create a new class called **Message.java**.

1.3. Define the **Message** class with a private **content** field and a constructor to initialize it.

1.4. Add a getter method (**getContent()**) to retrieve the content of the message.

Step 2: Create the **Sender** Class

2.1. Create a new class named **Sender.java**.

2.2. Define the **Sender** class with a method **createMessage** to create a message and a method **sendMessage** to send the message to a receiver.

2.3. In the **createMessage** method, instantiate a **Message** object and print a message indicating that the message has been created.

2.4. In the **sendMessage** method, print a message indicating that the message is being sent to the receiver and then invoke the **receiveMessage** method of the **Receiver** class.

Step 3: Create the **Receiver** Class

3.1. Create a new class named **Receiver.java**.

3.2. Define the **Receiver** class with a method **receiveMessage** to receive and process a message.

3.3. In the **receiveMessage** method, print a message indicating that the message has been received and display the content of the message.

Step 4: Create the Main Class

4.1. Create a new class named **MessagePassingExample.java**.

4.2. Define the **MessagePassingExample** class as the main class.

4.3. In the **main** method, create instances of **Sender** and **Receiver**.

4.4. Use the **Sender** instance to create a message and send it to the **Receiver**.

Step 5: Compile and Run

5.1. Compile all three classes (**Message**, **Sender**, and **Receiver**) by executing the appropriate commands in your IDE or using the command line.

5.2. Run the **MessagePassingExample** class.

5.3. Observe the output to verify that the message is created, sent, and received as expected

1.5 LEARN: EXPORT YOUR PROJECT

Please export your project as a ZIP file and save it in a safe place in one of your drives. See the following video as a guidance.

Section 2

2.1 LEARN: EXCEPTIONS IN JAVA

In Java, exceptions occur during the execution of a program that disrupts the normal flow of instructions. Exceptions are used to handle runtime errors and exceptional conditions that may arise during program execution.

2.2 LEARN: TRY-CATCH BLOCKS

Code that may raise an exception is placed within a try block.

Exception handling is done in the catch block, where specific types of exceptions can be caught and handled.

2.3 LEARN: EXCEPTIONS TYPES

Java has a hierarchy of exception classes. The most general type is Exception, and more specific types extend from it.

https://www.javatpoint.com/types-of-exception-in-java

2.4 LEARN: THROWING EXCEPTIONS

Exceptions can be explicitly thrown using the throw keyword.

In the following ExceptionExample class:

1. The first try-catch block demonstrates handling an **ArithmeticException** that might occur when dividing two numbers. An appropriate error message is displayed if the user attempts to divide by zero.

2. The second try-catch block demonstrates handling an **ArrayIndexOutOfBoundsException** when trying to access an element at an index outside the array's bounds.

```
    import java.util.Scanner;

    public class ExceptionExample {

        public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);

            // Example 1: ArithmeticException
            try {
                System.out.println("Enter a number: ");
                int num1 = scanner.nextInt();
                System.out.println("Enter another number: ");
                int num2 = scanner.nextInt();

                int result = num1 / num2;
                System.out.println("Result of division: " + result);
            } catch (ArithmeticException e) {
                System.out.println("Error: Division by zero is not
    allowed.");
            } catch (Exception e) {
                System.out.println("An unexpected error occurred.");
                e.printStackTrace();
            }

            // Example 2: ArrayIndexOutOfBoundsException
            try {
                int[] numbers = {1, 2, 3, 4, 5};
                System.out.println("Enter an index to get the
    corresponding number from the array: ");
                int index = scanner.nextInt();

                int value = numbers[index];
                System.out.println("Value at index " + index + ": " +
    value);
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Error: Index out of bounds. Please
    enter a valid index.");
            } catch (Exception e) {
                System.out.println("An unexpected error occurred.");
                e.printStackTrace();
            } finally {
                scanner.close();
            }
        }
    }
```

Please try the above codes in any Java IDE and provide inputs to see how Exception handling works.

Task: Please add another example, to catch **NullPointerException** when attempting to access the length of a null string (str). The catch block handles the exception and provides an appropriate error message.

2.5 LEAR: LOGGING IN JAVA

Logging in Java is a fundamental aspect of application development, providing a systematic way to record information about the program's execution. Java provides the **java.util.logging** package for logging purposes. Here's a brief explanation with simple examples:

1. Basic Logging:

The **Logger** class is the core component for logging in Java. You can create a logger using the class name or a custom name:

```java
import java.util.logging.Logger;

public class SimpleLoggingExample {
    private static final Logger logger =
Logger.getLogger(SimpleLoggingExample.class.getName());

    public static void main(String[] args) {
        logger.info("This is an information message.");
        logger.warning("This is a warning message.");
        logger.severe("This is a severe message.");
    }
}
```

Tip: To automatically set up a logger for your classes, you can simply do right click on any empty space inside the class and click on insert code and then select Logger.

2. Configuring Logging:

Java logging allows you to configure the logging behavior through a properties file or programmatically. Here's an example of configuring logging programmatically:

```java
import java.util.logging.ConsoleHandler;
import java.util.logging.Level;

public class Example {
    private static final Logger logger =
Logger.getLogger(Example.class.getName());

    public static void main(String[] args) {
        // Create a console handler
        ConsoleHandler consoleHandler = new ConsoleHandler();

        // Set the logging level for the handler
        consoleHandler.setLevel(Level.ALL);

        // Add the handler to the logger
        logger.addHandler(consoleHandler);

        // Log messages
        logger.info("This is an informational message.");
        logger.warning("This is a warning message.");
        logger.severe("This is a severe message.");
    }
}
```

3. Logging Exceptions:

```
import java.util.logging.Logger;

public class ExceptionLoggingExample {
    private static final Logger logger =
Logger.getLogger(ExceptionLoggingExample.class.getName());

    public static void main(String[] args) {
        try {
            // Some code that may throw an exception
            throw new RuntimeException("This is a sample exception.");
        } catch (Exception e) {
            logger.severe("An exception occurred: " + e.getMessage());
            logger.severe("Stack trace: ");
            for (StackTraceElement element : e.getStackTrace()) {
                logger.severe(element.toString());
            }
        }
    }
}
```

2.6 EXERCISE 2: IMPROVE THE EXERCISE 1 BY IMPLEMENTING LOGGING

Please import the exercise 1 that you exported as a ZIP file. Before importing the project, please unzip the file and then import it to your coding environment. Please refer to the provided video about how to import a project to NetBeans IDE.

After importing the project, please implement logging with appropriate logging levels and messages. You may use Logging instead of **System.out.println**.

Finally, compile and run the revised project and share your results with your instructor.

Please note that, you will need to include the following method in **MessagePassingExample** class:

```
private static void configureLogger() {
        try {
            // Configure global logging level
            Logger globalLogger = Logger.getLogger("");
            Handler[] handlers = globalLogger.getHandlers();

            for (Handler handler : handlers) {
                globalLogger.removeHandler(handler);
            }
            ConsoleHandler consoleHandler = new ConsoleHandler();
            consoleHandler.setLevel(Level.ALL);
            globalLogger.addHandler(consoleHandler);
            globalLogger.setLevel(Level.ALL);
        } catch (Exception e) {
            Logger.getGlobal().log(Level.SEVERE, "Error configuring
                                        logger: " + e.getMessage(), e);
        }
    }
```