I. Use Eclipse or Net bean platform and acquaint yourself with the various menus. Create a test project, add a test class, and run it. See how you can use auto suggestions, auto fill. Try code formatter and code refactoring like renaming variables, methods, and classes. Try debug step by step with a small program of about 10 to 15 lines which contains at least one if else condition and a for loop.

Using Eclipse:

1. Download and Install Eclipse:
   - Go to the Eclipse downloads page and download the Eclipse IDE for Java Developers.
   - Install Eclipse by following the on-screen instructions.
2. Create a Test Project:
   - Open Eclipse.
   - Go to File > New > Java Project.
   - Enter a project name (e.g., TestProject) and click Finish.
3. Add a Test Class:
   - Right-click on the src folder in your project.
   - Select New > Class.
   - Enter a package name (e.g., com.example) and a class name (e.g., TestClass).
   - Check the box to include the public static void main(String[] args) method.
   - Click Finish.
4. Write and Run a Test Program:
   - Open the TestClass.java file and add below code:

```
package com.example;
import java.util.Scanner;
public class TestClass {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter a number: ");
int num = scanner.nextInt();
if (num > 5) {
System.out.println("Number is greater than 5");
} else {
System.out.println("Number is 5 or less");
}
for (int i = 0; i < 5; i++) {
System.out.println("Iteration: " + i);
}
scanner.close();
}
}
```


   - Save the file and run it by right-clicking on the file and selecting Run As > Java Application.
5. Use Autosuggestions and Autofill:
   - Start typing code, and you will see Eclipse providing suggestions and autofill options. For example, type Sys and press Ctrl+Space to see suggestions for System.
6. Code Formatting:
   - Right-click on your Java file and select Source > Format to automatically format your code according to the default style settings.

7. Code Refactoring:
    - Right-click on a variable, method, or class name and select Refactor > Rename to rename it across the entire project.
8. Debug Step-by-Step:
    - Set a breakpoint by double-clicking on the left margin next to a line of code.
    - Right-click on the file and select Debug As > Java Application.
    - Use the debug perspective to step through the code, inspect variables, and control the execution flow.
9. Run:
    - To run the project Select run menu, click on the run (ctrl+F11).

Output:

Enter a number: 7
Number is greater than 5
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4

Characteristics of Object-Oriented Languages:

1. **Objects:** Objects are instances of classes that combine data (attributes or properties) and behaviors (methods or functions).
2. **Classes:** Classes are blueprints or templates that define the properties and behaviors common to all objects of that class.
3. **Encapsulation:** Encapsulation refers to bundling data and methods together within a class, hiding the internal workings of an object and exposing only the necessary interfaces.
4. **Inheritance:** Inheritance allows a class (subclass or derived class) to inherit properties and behaviors from another class (super class or base class), facilitating code reuse and promoting the DRY (Don't Repeat Yourself) principle.
5. **Polymorphism:** Polymorphism allows objects of different classes to be treated as objects of a common super class. It enables the same method to be used for different types of objects, providing flexibility and extensibility.
6. **Abstraction**: Abstraction is the concept of hiding the complex implementation details and showing only the necessary features. We'll use abstract classes to demonstrate abstraction.

II. Write a Java program to demonstrate the OOP principles and packages. [i.e., Encapsulation, Inheritance, Polymorphism and Abstraction]

Project Structure as below:
- `src`
  - `com.example.animal`
    - `Animal.java`
    - `Dog.java`
    - `Cat.java`
  - `com.example.main`
    - `MainClass.java`

`Animal.java` (Abstraction and Encapsulation)

```
package com.example.animal;
public abstract class Animal {
private String name;
private int age;
public Animal(String name, int age) {
this.name = name;
this.age = age;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public int getAge() {
return age;
}
public void setAge(int age) {
this.age = age;
}
public abstract void makeSound();
}
```

`Dog.java` (Inheritance and Polymorphism)

```
package com.example.animal;
public class Dog extends Animal {
public Dog(String name, int age) {
super(name, age);
}
@Override
public void makeSound() {
System.out.println("Woof Woof");
}
}
```

`Cat.java` (Inheritance and Polymorphism)

```
package com.example.animal;
public class Cat extends Animal {
public Cat(String name, int age) {
super(name, age);
}
@Override
public void makeSound() {
System.out.println("Meow Meow");
}
}
```

`MainClass.java`

```
package com.example.main;
import com.example.animal.Animal;
import com.example.animal.Dog;
import com.example.animal.Cat;
public class MainClass {
public static void main(String[] args) {
Animal myDog = new Dog("Buddy", 3);
Animal myCat = new Cat("Whiskers", 2);
System.out.println("Dog's Name: " + myDog.getName());
System.out.println("Dog's Age: " + myDog.getAge());
myDog.makeSound();
System.out.println("Cat's Name: " + myCat.getName());
System.out.println("Cat's Age: " + myCat.getAge());
myCat.makeSound();
}
}
```

Output:

```
Dog's Name: Buddy
Dog's Age: 3
Woof Woof
Cat's Name: Whiskers
Cat's Age: 2
Meow Meow
```

b) Java program that demonstrates the use of abstract classes and interfaces.

1. Create a New Java Project
    - Open Eclipse IDE.
    - Go to `File` -> `New` -> `Java Project`.
    - Enter the project name, for example, `AbstractInterfaceDemo`, and click `Finish`.

2. Create the Abstract Class
    - Right-click on the `src` folder of the project in the Project Explorer.
    - Go to `New` -> `Class`.
    - Name the class `Vehicle` and check the box for `abstract` (this will make it an abstract class). Click `Finish`. Write the below code in it.

`Vehicle.java`:

```
public abstract class Vehicle {
private String brand;
public Vehicle(String brand) {
this.brand = brand;
}
public String getBrand() {
return brand;
}
public abstract void start();
public abstract void stop();
```

}

3. Create the Interface
    - Right-click on the `src` folder again.
    - Go to `New` -> `Interface`.
    - Name the interface `FuelEfficient` and click `Finish`. Write the below code in it.

`FuelEfficient.java`:

```
public interface FuelEfficient {
double getFuelEfficiency(); // in kilometer per liter
}
```

4. Create Concrete Classes

Now, let's create a concrete class that extends the abstract class and implements the interface.
    - Right-click on the `src` folder again.
    - Go to `New` -> `Class`.
    - Name the class `Car`, check the box for `constructors from superclass`, `inherited abstract methods` and click `Finish`. Write the below code in it.

`Car.java`:

```
public class Car extends Vehicle implements FuelEfficient {
private double fuelEfficiency; // kilometer per liter
public Car(String brand, double fuelEfficiency) {
super(brand);
this.fuelEfficiency = fuelEfficiency;
}
@Override
public void start() {
System.out.println(getBrand() + " car is starting.");
}
@Override
public void stop() {
System.out.println(getBrand() + " car is stopping.");
}
@Override
public double getFuelEfficiency() {
return fuelEfficiency;
}
}
```

5. Main Class to Run the Program
    - Right-click on the `src` folder again.
    - Go to `New` -> `Class`.
    - Name the class `MainClass`, check the box for `public static void main(String[] args)`, and click `Finish`. Write the below code in it.

`MainClass.java`:

```
public class MainClass {
public static void main(String[] args) {
Car myCar = new Car("Toyota", 30.0);
myCar.start();
```

```
System.out.println("Fuel efficiency: " + myCar.getFuelEfficiency() + " KMPL");
myCar.stop();
}
}
```

6. Run the Program
    - Right-click on the `MainClass` class in the Project Explorer.
    - Select `Run As` -> `Java Application`.

Output:

```
Toyota car is starting.
Fuel efficiency: 30.0 KMPL
Toyota car is stopping.
```

III. Write a Java program to handle checked and unchecked exceptions. Also, demonstrate the usage of custom exceptions in real time scenario.

Project Structure as below:

- `src`
  a. `com.example.exception`
  i. `CustomException.java`
  ii. `ExceptionDemo.java`

Checked and Unchecked Exceptions:

Checked Exception - Checked exceptions must be either caught or declared in the method signature using `throws`.
Examples : `IOException` and `SQLException`.
Unchecked Exception - Unchecked exceptions do not need to be declared or caught.
Examples : `NullPointerException` and `ArithmeticException`.
Custom Exception - Custom exceptions can be created by extending the `Exception` class (for checked exceptions) or `RuntimeException` class (for unchecked exceptions).

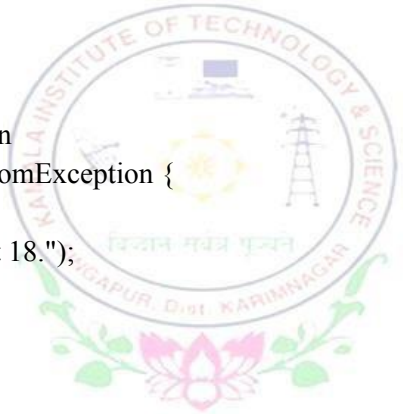
`CustomException.java` (Custom Checked Exception)

```
package com.example.exception;
public class CustomException extends Exception {
public CustomException(String message) {
super(message);
}
}
```

`ExceptionDemo.java`

```
package com.example.exception;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class ExceptionDemo {
public static void main(String[] args) {
// Demonstrating unchecked exception (ArithmeticException)
try {
```

```
int result = divide(10, 0);
} catch (ArithmeticException e) {
System.out.println("Caught an unchecked exception: " + e.getMessage());
}
// Demonstrating checked exception (IOException)
try {
readFile("nonexistentfile.txt");
} catch (IOException e) {
System.out.println("Caught a checked exception: " + e.getMessage());
}
// Demonstrating custom checked exception
try {
validateAge(15);
} catch (CustomException e) {
System.out.println("Caught a custom checked exception: " + e.getMessage());
}
}
// Method to demonstrate unchecked exception
public static int divide(int a, int b) {
return a / b; // This will throw ArithmeticException if b is 0
}
// Method to demonstrate checked exception
public static void readFile(String filename) throws IOException {
BufferedReader reader = new BufferedReader(new FileReader(filename));
String line;
while ((line = reader.readLine()) != null) {
System.out.println(line);
}
reader.close();
}
// Method to demonstrate custom checked exception
public static void validateAge(int age) throws CustomException {
if (age < 18) {
throw new CustomException("Age must be at least 18.");
}
}
}
```



Output:

Caught an unchecked exception: / by zero
Caught a checked exception: nonexistentfile.txt (The system cannot find the file specified)
Caught a custom checked exception: Age must be at least 18.

IV. Write a Java program on Random Access File class to perform different read and write operations.

```
import java.io.IOException;
import java.io.RandomAccessFile;
```

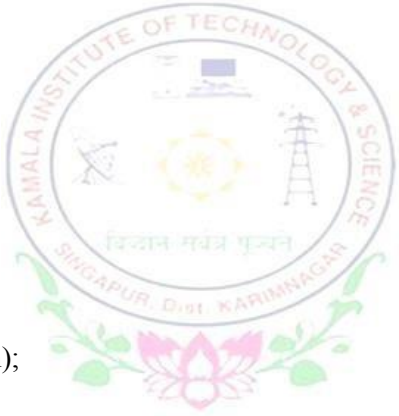
```
public class WriteUTFExample {
public static void main(String[] args) {
try {
// Open the file in "rw" (read/write) mode
RandomAccessFile file = new RandomAccessFile("example.txt", "rw");
// Write a string to the file using writeUTF (Unicode Transformation Format)
file.writeUTF("Hello, World!");
// Seek to the beginning of the file to read the string back
file.seek(0);
// Read the string using readUTF
String message = file.readUTF();
// Print the read string
System.out.println("Read from file: " + message);
// Close the file
file.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

Output:

Read from file: Hello, World!

b) Program that demonstrates how to copy the content of one file to another using file I/O operations.

- BufferedReader: Reads text from a file efficiently, line by line.
- BufferedWriter: Writes text to a file efficiently.
- FileReader: A convenience class for reading character from files.
- FileWriter: A convenience class for writing character to files.




```
import java.io.*;
public class FileCopier {
public static void main(String[] args) {
// Paths to source and destination files
String sourceFilePath = "source.txt";
String destinationFilePath = "destination.txt";
// Create File objects
File sourceFile = new File(sourceFilePath);
File destinationFile = new File(destinationFilePath);
// Perform the file copy
try (BufferedReader reader = new BufferedReader(new FileReader(sourceFile));
BufferedWriter writer = new BufferedWriter(new FileWriter(destinationFile))) {
String line;
while ((line = reader.readLine()) != null) {
writer.write(line);
writer.newLine(); // Ensure lines are separated correctly
```

```
}
System.out.println("File copied successfully!.");
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

Output:

File copied successfully!.