

Programming Final Assignment

Spring 2022/2023

A grey logo on a black background

Description automatically generated with low confidence

19 June 2023

Instructors: Dr. Huthaifa AL Omari / Eng. Mostafa Smadi

Student: Raneem Yahya Sa’deh

Student ID: 22210016

**P-R-O-G-R-A-M-M-I-N-G**



Course Name: Programming

HTU Course No: 40201100

BTEC Unit No: H/618/7388



# Programming Paradigm:

It is the main approach or style of programming that help us to design, structure, and solve problems in programming. It encompasses a collection of practices, principles, and concepts that deeply affect our coding mindset. There are 3 programming paradigms will be discussed in this report:

* Object-Oriented Programming (OOP)
* Procedural Programming
* Event-Driven Programming

Each paradigm has its unique characteristics and focuses on different aspects of program organization and execution.

## Object-Oriented Programming (OOP):

**A picture containing text, circle, screenshot, font

Description automatically generated**

Object-oriented programming is a core of Java Programming, and it improves code readability and reusability by defining a Java program efficiently.

It is used for designing a program using classes and objects. OOPs, can also be characterized as data controlling for accessing the code.

Abstraction, encapsulation, inheritance, and polymorphism are the four main principles of object-oriented programming.

This part will discuss the foundation concepts of the OOP in Java:

* Objects
* Classes
* Abstraction
* Inheritance
* Polymorphism
* Encapsulation

### Objects:

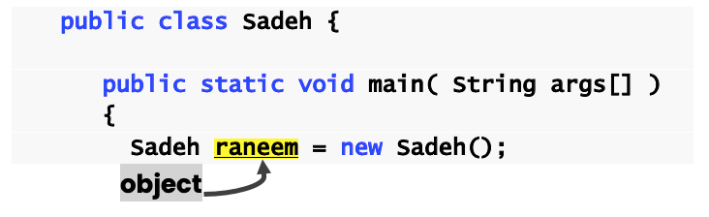


Figure - Code example of Objects

These objects always correspond to things found in the real world. It contains addresses and takes up some space in memory. For instance, a book, a cat ...

Objects are always called instances of a class. Objects have state (attributes or properties) and behavior (methods or functions) associated with them. They interact with each other by sending messages and invoking methods.

### Classes:

A class is a blueprint or template for creating objects. It defines the structure and behavior that objects of the class will have. All class objects should have basic class properties. It includes attributes, values, and methods that may be used by the object.

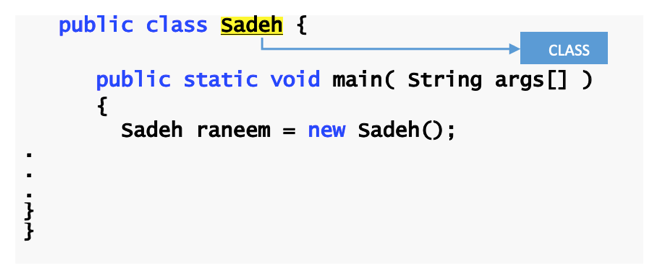


Figure - Code example of Class

A class declaration consists of[[1]](#footnote-1):

* Modifiers--> These can be public or default access.
* Class name--> The first letter must be capitalized.
* Superclass--> A class can only extend (subclass) one parent.
* Interfaces--> I make more than one Interface (Event Driven) in one class.
* Body--> The body of the class must start and end with {}.

### Abstraction:

Abstraction is a technique that only reveals the information that is relevant and hides the rest. We could say that the main objective of abstraction is data concealment. By selecting data from a large pool of data to display the necessary information, abstraction reduces programming labor and complexity.

A picture containing text, font, screenshot, handwriting

Description automatically generatedClasses and methods that are abstract are also present. An abstract class is one that specifies one or more abstract methods. An abstract method is one that has a method specification but no implementation. Once we have modeled our object using data abstraction, the same pieces of data can also be used in multiple applications, including abstract classes, generic types of behaviors, and object-oriented programming hierarchy.

Figure - Code example of using Abstact in Java

Keyword [**abstract**]:

* Use to declare a class [**abstract**]
* Use to declare a method [**abstract**]

Abstract classes normally contain one or more abstract methods. All concrete subclasses must override all inherited abstract methods.

### Inheritance:

One way for an item to obtain or inherit another object's properties is through inheritance, which also permits hierarchical classification. The concept behind this is that we may build new classes on top of older classes; in other words, when you inherit from an older class, you can use its fields and methods again. The parent-child relationship is represented by inheritance. (Where children always inherit from their parents)

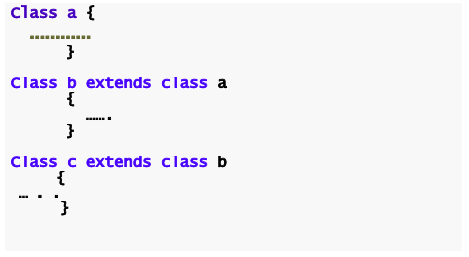


Figure - This code shows that the class “c” inherits from class “b” that inherits from class “a” which is the parent.

### Polymorphism:

The Java programming language depends extensively on polymorphism, a basic idea in object-oriented programming. It offers flexibility, reusability, and extensibility in program design by enabling objects from different classes to be considered as objects of a common superclass or interface.

Method overloading and overriding are the two main strategies used in Java to implement polymorphism.

Overriding:

Overloading:



Figure - This code shows two methods with the same name but each of them has different signs that make them different from each other.

Multiple methods in the same class may have the same name but distinct parameters thanks to method overloading. With different input arguments or return types but equivalent functionality, it lets a class to have several methods. Java bases its decision on the quantity, kind, and ordering of the arguments given during the method call.

### Encapsulation:

A picture containing text, screenshot, font

Description automatically generatedOne of the OOPs principles is encapsulation, which is the process of combining data and code into a single entity while protecting both from outside intervention and misuse. The data is kept secret from other classes during this procedure and is only accessible through the methods of the current class. Thus, it is sometimes referred to as data concealing. Encapsulation serves as a barrier that keeps others from accessing the code and data. These are managed via a well laid out interface.

By designating the variables as private and making the "setter" and "getter" methods public, encapsulation can be accomplished. The fields of a class are turned read-only or write-only during encapsulation. This method also improves reusability. Encapsulated code is also easy to test for unit testing.

### The benefits and drawbacks of Object-Oriented Programming:

|  |  |
| --- | --- |
| Benefits | It allows for the creation of reusable and self-contained objects, making code maintenance and collaboration more manageable. |
| Drawbacks | OOP can introduce complexity, especially for beginners or when dealing with overly intricate class hierarchies. |

## Procedural:

A picture containing text, screenshot, font, rectangle

Description automatically generatedProgramming methodology known as **procedural programming** emphasizes partitioning a program into several processes or functions. It executes in a linear, step-by-step manner, with each procedure carrying out a distinct function and being callable or invoked as required.

**Variables** are used in procedural programming to store and control data. Within the procedures in which they are declared, variables have certain data types and usage boundaries. The use of global variables or parameters allows data to be transmitted between operations.

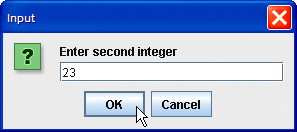
Because of procedural abstraction, complex operations **can be broken down into smaller**, simpler procedures. By breaking the program's complexity into smaller, more manageable pieces, this aids in controlling program complexity.

Procedural programming **lacks** the concepts of inheritance and polymorphism, which are key features of object-oriented programming. Instead, it focuses on procedural decomposition and reuse.

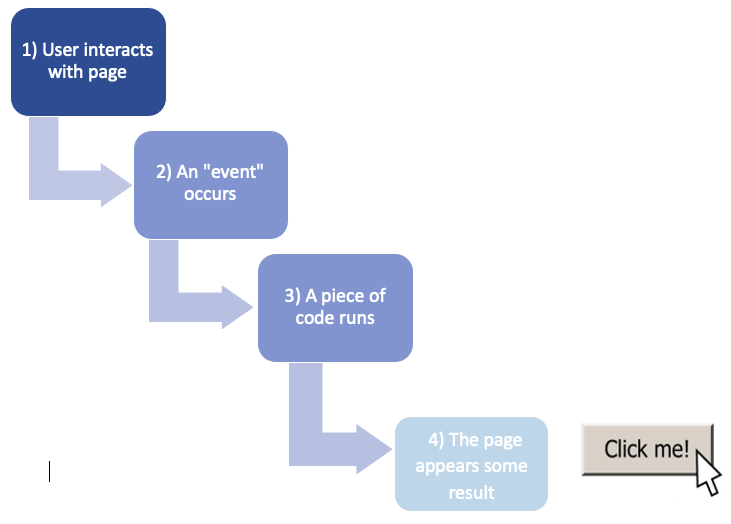
### The benefits and drawbacks of Procedural:

|  |  |
| --- | --- |
| Benefits | Procedural programming is generally simpler and easier to understand for small-scale programs. It follows a linear flow of execution, making it more straightforward to write and debug code. |
| Drawbacks | Procedural programming lacks strong modularity and encapsulation since it relies on global variables and functions. |

## Event-Driven:

A programming paradigm known as "**event-driven programming**" focuses on designing software systems around events and the event handlers that go along with it. According to this paradigm, program flow is influenced by many events, such as user interaction, sensor input, or communications from other components.

The idea of events is central to event-driven programming. An event is a signal or indication that something has occurred inside or externally to the system. It might be a mouse click, a button press, the arrival of a network packet, or something else entirely.

**Event handlers**, which are functions or methods created particularly to react to particular occurrences, are essential to event-driven programming. The corresponding event handler is triggered or called when an event happens.

Event-driven programming entails attaching event handlers to certain events or event sources. This creates a link between events and the handlers that correspond to them. The proper handler is automatically called whenever an event occurs.

Graphical **user interfaces** (GUIs) and interactive applications are frequently created using event-driven programming. User activities, including mouse clicks, and keyboard input produce events that cause the appropriate event handlers to either change the user interface or carry out particular tasks.

### The benefits and drawbacks of Event-Driven:

|  |  |
| --- | --- |
| Benefits | It allows for creating responsive and interactive user interfaces. It enables programs to respond to various events such as button clicks which enhancing the user experience. |
| Drawbacks | Managing and coordinating multiple events, callbacks, and maintaining program state can become challenging, leading to potential issues. |

## Good Algorithm

According to the criteria given, a good algorithm demonstrates a number of characteristics. First, it must be unambiguous and straightforward, which means that every step of the algorithm must be simple to understand and have just one possible interpretation. Second, it should contain inputs that are clearly defined and state the values that are expected as input. Thirdly, it must have outputs that are precisely stated, stating what information or outcomes will be generated by the algorithm. Fourth, the algorithm needs to be finite, which means it won't continue indefinitely or encounter any other comparable problems. Fifth, it must be practical, generic, and able to be implemented with already available resources without relying on a future technology.

Last but not least, the algorithm should be language-neutral, composed of straightforward instructions that may be used in any programming language and yet result in the desired result.

We assessed the following Java code if it meets the explained GOOD AlGORITHM criteria:

package final\_assignment;

import java.util.Scanner;

public class GameCompanyProcedural

{

public static void main(String[] args){

Scanner input = new Scanner(System.in); //using the Scanner class to let the user enter the value of the variable

int age ;

String name ;

String work ;

String ssn;

System.out.println("Enter your name : ");

name = input.nextLine();

System.out.println("Enter your age : ");

age = input.nextInt();

System.out.println("Enter your ssn : ");

ssn = input.nextLine();

System.out.println("Choose what is your work A) Manager B) Developer : ");

work = input.nextLine();

System.out.println(isManager("manager"));

System.out.print("your name is : "+name+"your age is : "+age+"your ssn is : "+ssn);

}

public static boolean isManager(String work) {

if(work == "man") {

System.out.println("You are working as a Manager and you will receive 3000 JD every month");

return true;

} else if(work == "dev"){

System.out.println("You are working as a dev and you will receive 3000 JD every month");

return false;

}else {

System.out.println("your input isnt correct please chose from (man) or (dev)");

return false;

}

}

}

Here is the assessment of the provided code against each criterion:

|  |  |
| --- | --- |
| Criterion | Assessment |
| Clear and Unambiguous | The algorithm is reasonably unambiguous and clear. The procedures are simple to follow and understand. To increase clarity, there are a few tweaks that can be performed. For instance, the code can have comments that describe the purpose of specific actions and more descriptive variable names. |
| Well-Defined Inputs | The method specifies in detail the user inputs that are needed, including name, age, ssn, and place of employment. Using the Scanner class, it asks the user for these inputs. |
| Well-Defined Outputs | The algorithm generates statements for System.out.println() that display the user's name, age, and ssn. The isManager() method also generates messages based on the user's selected work. The isManager() method's output messages, however, do not reveal the true worth of the input labor. |
| Finite-ness | The lack of loops or recursive calls, which could result in unlimited execution, gives the algorithm the appearance of being finite. It uses user inputs and outputs to carry out the processes in a linear fashion. |
| Feasible | Due to the use of fundamental input/output operations and conditional statements, the approach is straightforward and practicable. Because it doesn't rely on any cutting-edge or upcoming technology, it can be implemented effectively using the resources at hand, and I used the input and output many times in my code. |
| Language Independent | The algorithm is largely agnostic to languages. With modest adjustments, the algorithm's logic can be implemented in a variety of programming languages. However, the existing code contains particular Java terminology and libraries, like the Scanner class, which could need to be modified when the technique is implemented in a new language. |

# Explaining the Project’s codes

## Procedural:

Let's break down the code step by step:

1. The main method, which serves as the program's entry point, begins the code. Although it can take an array of strings as a parameter, args, this instance does not use it.
2. In order to read user input, the program creates a Scanner object called input. The Java standard library's Scanner class enables reading input from a variety of sources, in this case, the standard input (System.in).



1. Several variables, including age (an integer), name (a string), work (a string), and ssn (a string to store the social security number), are declared to hold the user's input.

A picture containing text, font, screenshot, number

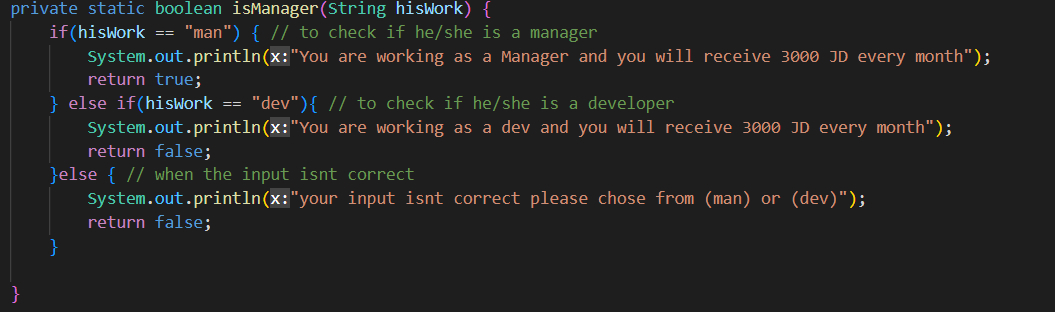
Description automatically generated

1. The program uses System.out.println() to prompt the user to enter their name before reading the input using input.and adds it to the name variable using nextLine().
2. In a similar manner, the program requests the user's age and reads it from the input.nextInt().
3. After that, the program uses System.out.println() to ask the user to enter their social security number (SSN).
4. Before reading the SSN, use nextLine() after reading the age to consume the newline character.
5. The program uses input to read the SSN input.and transfers it to the ssn variable using nextLine().
6. The program then displays a message with the options "A" and "B" using System.out.println(), asking the user to select their work designation. Using input, it reads the input.and adds it to the work variable using nextLine().

A screenshot of a computer code

Description automatically generated with low confidence

1. The work variable is passed as an argument when the isManager method is called by the program. The method determines whether "man" or "dev" is the work designation and then returns a boolean value indicating whether or not the person is a manager.



1. The program uses System.out.println() to display the appropriate message based on the isManager function's return value.
2. Finally, the program prints the user's name, age, and SSN using **System.out.print()**.
3. The hisWork String parameter, which denotes the work designation, is required by the isManager method. The == operator is used to compare the value of hisWork with "man" and "dev" and to determine whether the references are equal.
4. If the work designation is "man," a message stating that the person is a manager and receives 3000 JD (Jordanian Dinar) per month is printed inside the isManager method. Indicating that the person is a manager, it returns true.
5. If the job title is "dev," a message stating that the person is employed as a developer and receives 3000 JD per month is printed. To demonstrate that the user is not a manager, it returns false.
6. If the user enters a work designation other than "man" or "dev," a message stating that the input is incorrect and requesting that they select "man" or "dev" is printed. The result is false.

A screen shot of a computer screen

Description automatically generated with low confidence

## EvenDriven:

* String Initialization:



- A string variable named **welcome** is initialized with the value "Welcome to Game Company".

* Displaying Welcome Message:

A picture containing text, font, screenshot, graphics

Description automatically generated

- A dialog box is displayed using **JOptionPane.showMessageDialog**, showing the contents of the **welcome** variable. The **null** parameter means the dialog box is displayed in the center of the screen.

* User Input: Name, Age, and Major:

A screen shot of a computer program

Description automatically generated with low confidence

- Three dialog boxes are displayed sequentially, using **JOptionPane.showInputDialog**. The user is prompted to enter their name, age, and major. The entered values are stored in the corresponding variables **name**, **ageString**, and **majorString**.

* Parsing Age as an Integer:

A picture containing text, font, screenshot, typography

Description automatically generated

- The **ageString** variable is converted to an integer using **Integer.parseInt()** and stored in the **age** variable.

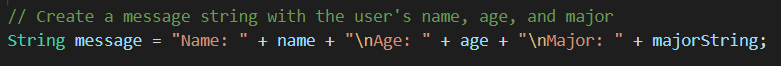
* Handling Major Selection:

A screen shot of a computer code

Description automatically generated with low confidence

- Two variables, **developer** and **managerString**, are initialized as null. If the **majorString** is equal to "developer" or "manager", the corresponding variable (**developer** or **managerString**) is assigned to **majorString**.

* Constructing Message String:



- A string variable named **message** is created by concatenating the entered name, age, and major.

* Creating GameCompanyEventDriven Object:



- An instance of the **GameCompanyEventDriven** class is created.

* Displaying Final Message:

A picture containing text, font, screenshot, graphics

Description automatically generated

- The **message** string is displayed in a dialog box using **JOptionPane.showMessageDialog**.

## OOP:

### Emp:

1. attributes:
   * **age** (private int): Represents the age of the employee.
   * **name** (private String): Represents the name of the employee.

A picture containing text, font, screenshot, number

Description automatically generated

1. Constructors:
   * **emp()**: This is the default constructor, which does not take any parameters. It doesn't perform any specific actions.
   * **emp(int age, String name)**: This is a parameterized constructor that accepts an age and name for the employee. It initializes the **age** and **name** variables with the provided values.

A picture containing text, screenshot, font

Description automatically generated

1. Setter Methods:
   * **setAge(int age)**: Sets the value of the **age** variable to the specified age.

A picture containing text, font, screenshot

Description automatically generated

* + **setName(String name)**: Sets the value of the **name** variable to the specified name.

A picture containing text, screenshot, font

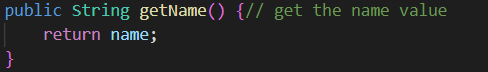
Description automatically generated

1. Getter Methods:
   * **getAge()**: Retrieves the value of the **age** variable.

A picture containing text, font, screenshot, handwriting

Description automatically generated

* + **getName()**: Retrieves the value of the **name** variable.



1. Additional Method:
   * **sayHello()**: This method does not take any parameters and does not return any value. It simply prints "Hello" to the console.

A picture containing text, font, screenshot, graphics

Description automatically generated

The class ensures proper control over accessing and changing these variables by encapsulating the age and name variables and offering setter and getter methods. Saying "Hello" to the console is a basic greeting function offered by the sayHello() method.

### developer:

Attributes:

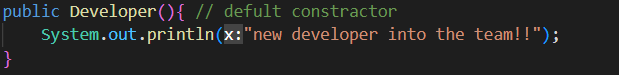
1. "framework": This private instance variable represents the framework that the developer is using. It is of type String.
2. "devExperience": This private instance variable represents the years of experience the developer has. It is of type int.

A picture containing font, text, screenshot, graphics

Description automatically generated

Constructors:

1. Default Constructor: This constructor does not take any parameters. When an object of the "Developer" class is created using this constructor, it will print "new developer into the team!!" to indicate the addition of a new developer to the team.



1. Parameterized Constructor: This constructor takes four parameters: "name" (String), "age" (int), "devExperience" (int), and "framework" (String). It calls the superclass constructor "emp(age, name)" using the "super" keyword to initialize the inherited attributes. It also assigns values to the "devExperience" and "framework" attributes. Similar to the default constructor, it also prints "new developer into the team!!" to indicate the addition of a new developer to the team.

A screen shot of a computer

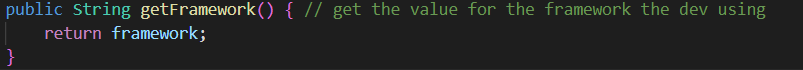
Description automatically generated with low confidence

Methods:

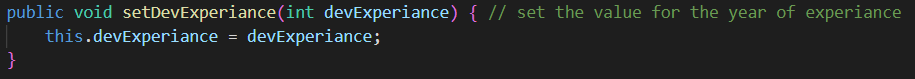
1. "setFramework": This method sets the value of the "framework" attribute using the provided parameter.



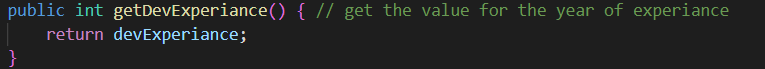
1. "getFramework": This method returns the value of the "framework" attribute.



1. "setDevExperience": This method sets the value of the "devExperience" attribute using the provided parameter.



1. "getDevExperience": This method returns the value of the "devExperience" attribute.



1. "sayHello": This method overrides the "sayHello" method from the superclass. It prints "Hello from the developer" to greet the developer.

A picture containing text, font, screenshot

Description automatically generated

### Manager:

Attributes:

1. **layer**: This attribute represents the layer or level of the manager within the organizational hierarchy.
2. **manExperiance**: This attribute stores the number of years of experience the manager has.

A picture containing font, text, screenshot, typography

Description automatically generated

Constructor:

1. The default constructor **Manager()** is responsible for creating a new instance of the "Manager" class. It outputs a message to indicate that a new manager has joined the team.

A picture containing text, font, screenshot, graphics

Description automatically generated

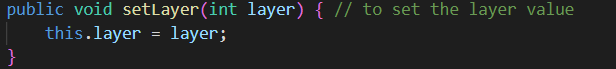
1. The parameterized constructor **Manager(int age, String name, int layer, int manExperiance)** initializes the inherited attributes from the "emp" class (age and name) using the **super(age, name)** statement. It also initializes the "layer" and "manExperiance" attributes with the provided values. It outputs a message to indicate that a new manager has joined the team.

A screen shot of a computer

Description automatically generated with low confidence

Methods:

1. **setLayer(int layer)**: This method sets the value of the "layer" attribute.

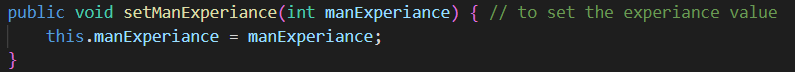


1. **getLayer()**: This method returns the value of the "layer" attribute.

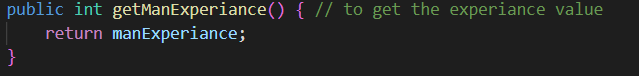
A picture containing text, font, screenshot

Description automatically generated

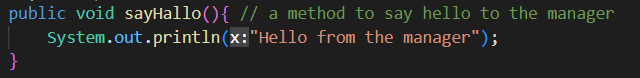
1. **setManExperiance(int manExperiance)**: This method sets the value of the "manExperiance" attribute.



1. **getManExperiance()**: This method returns the value of the "manExperiance" attribute.



1. **sayHello()**: This method outputs a message to greet the manager.



# What is an Error in Java?

An error is an illegal statement or operation a programmer makes in code and it causes the program to behave abnormally and restricts the application from executing. Errors usually remain undetected until the program is compiled or executed. Some may cause the program's termination or may inhibit the program from getting compiled or executed.

When can an error happen?

* Java programming language has a set of rules to write the program. An error can happen if the programmer doesn't follow these rules.
* When a programmer performs an operation that is not intended because of a wrong idea or concept.

## A picture containing text, screenshot, font, diagram Description automatically generatedTypes of Error in programming

In Java, there are three types of errors:

* Syntax Errors
* Runtime Errors
* Logical Errors

### Syntax Errors:

Syntax errors happen when syntactical problems happen in a Java program because of incorrect use of Java syntax, these errors are found by the Java compiler at compile time of the program. Syntax errors are easy to spot because the Java compiler finds them for you. After running the code an error message will be displayed on the output screen.

And it happens when:

* You miss a semicolon.
* Miss brackets.
* Misspelled keywords.
* Use undeclared variables.
* Class not found.
* You miss a double quote in String.

Example:

Class Raneem {

Public static void main(String[] args) {

int a = 3;

int b = 4

}

}*//Missing semicolon*

### Runtime Errors:

Runtime errors happen when the program has successfully compiled without giving any errors and creating a ".class" file. However, the program does not execute properly. These errors are detected at the time of execution of the program. These runtime errors are called exceptions and they terminate the program abnormally, giving an error statement. We can use exception-handling techniques in Java to handle these runtime errors. In exception handling, the piece of code the programmer thinks can produce the error is put inside the try block and the programmer can catch the error inside the catch block.

And it happens when:

* When you divide an integer by zero.
* Trying to store a value into an array that is not compatible type.
* Trying to access an element that is out of range of the array.
* Passing an argument that is not in a valid range or valid value for a method.
* Striving to use a negative size for an array.

Example:

Class Raneem {

Public static void main(String[] args) {

int[] arr = new int[5];

try {

arr[8] = 89; *//The number is out of the range for the array.*

} catch (Exception e) {

System.out.println("There will be a Runtime error!");

}

}

}

### Logical Errors:

Logical errors are the hardest to identify and correct. They are the hardest to detect because they aren't identified by the Java compiler or by the JVM. The programmer is entirely responsible for them to correct them by himself. The program with a logical error will get compiled and executed successfully but will not give the expected result that the programmer wants to see. Logical errors can be detected by application testers when they compare the actual result with the program's expected result.

Example:

Class Raneem {

Public static void main(String[] args) {

for (int i = 0; i <= 10; i++) {

if (i / 2 == 0) {

System.out.println(i);

}

}

}

}

**The Expected output should be : But the output will be :**

0 0

2 1

4

6

8

10

# Debug:

Software development must include the debugging process, which seeks to locate and fix faults, bugs, and other problems in a program's source code. To find and address issues, it entails methodically examining the code and the behavior of the program. An overview of the Java debugging process and how it helps create programs that are more reliable and secure is given below:

Finding the issue or problem that needs to be fixed is the first step in debugging. Error messages, unexpected behavior, crashes, or inaccurate output can all accomplish this. To adequately examine a problem, the developer must be able to consistently reproduce it.

The developer makes an effort to reliably reproduce the issue once it has been found. To do this, the application might need to be run with particular input, data, or circumstances in order to cause a problem. Reproducibility is essential for correctly identifying and analyzing the issue. Examining the code associated with the problematic location comes next. This entails checking over the pertinent source code parts for logical mistakes, improper syntax, or any other probable sources of the issue. Breakpoints and stepping during code execution are capabilities offered by debugging tools like integrated development environments (IDEs) like Eclipse, which help with code inspection.

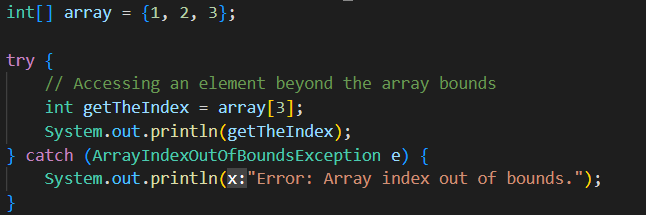
Breakpoints are defined locations in the code where the running of the program is momentarily stopped to permit inspection. Developers can see the state and variable values of the program at various places by carefully adding breakpoints. This aids in identifying the precise location of the issue and in comprehending the factors or circumstances that contribute to it.

Developers examine the state of the software during runtime when debugging to learn how the variables and data are changing. To comprehend the operation of the program and spot potential issues, they can monitor variable values as well as any error or exception messages. This analysis aids in identifying the problem's underlying causes.

The developer can add the required fixes to the code after identifying and analyzing the issue. To remedy vulnerabilities, this may entail fixing logical mistakes, appropriately resolving exceptions, or bolstering security measures.

The developer must extensively test the modified code after applying the fix to make sure the issue has been fixed and that no new problems have been caused by it. Regression testing, running test cases, and comparing the program's behavior to anticipated results are all part of this process.

## Debugging techniques:

1. Code comprehension: Comments give additional context and explanations about the goal, logic, or anticipated behavior of the code. When debugging, comments make it simpler to identify potential errors or inconsistencies by assisting you in understanding the purpose behind particular lines of code.
2. Disable Code Segments: On occasion, you might think that a specific piece of code is the source of the problem. You can temporarily disable it by commenting out that section and see if the issue still exists. This aids in isolating the problematic code and identifying the bug's primary cause.
3. Exception handling with try and catch: You can handle any exceptions that come up during execution by enclosing a suspicious or error-prone section of code in a try block. The program flow is moved to the corresponding catch block when an exception is thrown, enabling you to gracefully handle the exception. By doing this, the program is kept from ending abruptly and given the chance to record the error, show an error message, or take corrective action.
4. A picture containing text, screenshot, display, software

   Description automatically generatedError identification using try and catch: Whenever an exception is caught, information about the kind of exception that occurred and where in the code it occurred is typically provided. This assists in identifying the precise error and its underlying cause. The error's cause, as well as the values of variables and the order in which method calls were made leading up to it, can be determined from the exception message and stack trace.
5. Print Statements: Adding print statements at different points in the code helps to understand the flow of execution and the values of variables. You can print variable values, function calls, and intermediate results to track the program's behavior.
6. A picture containing text, screenshot, software, multimedia software

   Description automatically generatedExamining Variable Values with breakpoint: You can check the values of variables in the current scope when the program execution is halted at a breakpoint. This aids in determining the program's current state and reveals any unexpected or flawed values. Expressions can be evaluated to see if they produce the desired outcomes.
7. Stepping Through the Code: After the program has been stopped at a breakpoint, you can step through the code using debugging commands. Stepping over lines (running the next line without moving on to function calls), into functions to observe their behavior, or out of a function to return to the caller are all examples of this. This makes it easier to track the execution path and spot any problems or strange behaviors.

# Coding Standards

***Coding standards*** are a set of guidelines that reputable software development companies expect their programmers to conform to. According to their organization's needs and the kinds of software they create, they typically create their coding standards and rules. Maintaining coding standards is crucial for programmers; otherwise, the code would be rejected during code review. ***A code style that is too challenging to grasp should be avoided:***

The code needs to be **simple to understand**. Maintenance and bug fixing is difficult and expensive because of the sophisticated code. Avoid using the same identifier more than once; instead, give each variable a meaningful name that describes its function. If one identifier is used for several reasons, this is not possible, and the reader may become confused as a result. Furthermore, it makes future improvements more challenging.

***Purpose of Having Coding Standards:***

Coding standards are a set of guidelines that reputable software development companies expect their programmers to conform to. According to their organization's needs and the kinds of software they create, they typically create their coding standards and rules. Maintaining coding standards is crucial for programmers; otherwise, the code would be rejected during code review.

***Code must have thorough documentation:***

The code should have **clear comments** to make it easier to understand. The code is easier to understand when there are comments regarding the statements.

Functions shouldn't be very long because they are incredibly challenging to comprehend. To accomplish minor tasks, functions should be tiny enough to handle them, and large functions should be divided into smaller ones.

1. Naming Conventions:

• Class names: Use PascalCase (also known as UpperCamelCase) for class names. For example: MyClass.

• Method and variable names: programmers uses camelCase for method and variable names. For example: myMethod, myVariable.

• Constant names: Use uppercase letters with underscores For example: GAME\_TYPE.

2. Indentation and Braces:

• We use consistent indentation (usually 4 spaces or a tab) to improve code readability.

• Also Placing opening braces { on the same line as the corresponding statement or declaration.

• Use proper spacing around operators and between elements for readability.

3. Comments:

• Use comments to explain the purpose, functionality, or any important details of the code.

• Include inline comments sparingly, focusing on explaining complex or non-obvious parts of the code.

• Write clear and concise comments that enhance understanding without h redundant.

## Examples from the submitted codes:

|  |  |
| --- | --- |
| Space after the comma between function arguments: |  |
| Proper indentation of nested blocks: | A screen shot of a computer program  Description automatically generated with low confidence |
| Braces on a new line and code after braces on a new line. | A screen shot of a computer program  Description automatically generated with low confidence |
| Using clear names that correspond to their use. | A screenshot of a computer  Description automatically generated with low confidence |
| Indentation at the beginning and end of each block. | A screen shot of a computer code  Description automatically generated with medium confidence |

**General Examples:**

* + 1. Space after the comma between function arguments:

public class Example {

public static void main (String[] args, int value) {

**………..**

}

}

* + 1. Proper indentation of nested blocks:

public class Example {

public static void main(String[] args) {

if (condition) {

**………..**

if (nestedCondition) {

**………..**

}

}

}

}

* + 1. Indentation at the beginning and end of each block:

public class Example {

public static void main(String[] args) {

if (condition) {

// Code here

} else {

**………..**  
}

* + 1. Braces on a new line and code after braces on a new line:

public class Example {

public static void main(String[] args)

{

if (condition)

{

**………..**

}

else

{

}

}

}

* + 1. Using clear names that correspond to their use:

public class Example {

public static void main(String[] args, int value) {

int firstName;

int lastName;

System.out.println(“enter you first and last name :”)

Scanner input = new Scanner(nextln);

}

# References

* Bose, Shreya. “Coding Standards and Best Practices to Follow.” BrowserStack, 2 Feb. 2021, [www.browserstack.com/guide/coding-standards-best-practices](http://www.browserstack.com/guide/coding-standards-best-practices). Accessed 19 June 2023.
* Heusser, Matt. “What Is Debugging?” SearchSoftwareQuality, [www.techtarget.com/searchsoftwarequality/definition/debugging](http://www.techtarget.com/searchsoftwarequality/definition/debugging). Accessed 19 June 2023.
* Mishra, Mudit. “Types of Errors in Java.” Scaler Topics, 8 Oct. 2022, [www.scaler.com/topics/types-of-errors-in-java/](http://www.scaler.com/topics/types-of-errors-in-java/). Accessed 19 June 2023.
* Tarelho, Otavio. “Event-Driven Architecture and Its Application in Java - Part 1.” Blog.avenuecode.com, 15 Jan. 2020, blog.avenuecode.com/event-driven-architecture-and-its-application-in-java-part-1. Accessed 19 June 2023.

1. https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/ [↑](#footnote-ref-1)