PROJECT REPORT

ON

SNAKE GAME

IN JAVA

Submitted By:

Ramanpreet Kaur

Raj Sharma

Pallavi Bharti

Parneet Singh

Rishi Suri

Vishal Yadav

ABSTRACT

Snake game is a computer action game, whose goal is to control a snake to move and collect food in a map. In this paper we develop a controller based on movement rating functions considering smoothness, space, and food. Scores given by these functions are aggregated by linear weighted sum, and the snake takes the action that leads to the highest score. To find a set of good weight values, we apply an evolutionary algorithm. We examine several algorithm variants of different crossover and environmental selection operators.

In the game of Snake, the player uses the arrow keys to move a "snake" around the board. As the snake finds food, it eats the food, and thereby grows larger. The game ends when the snake either moves off the screen or moves into itself. The goal is to make the snake as large as possible before that happens.

ACKNOWLEDGEMENT

I take this opportunity to express my sincere gratitude to the Principal, Chandigarh Engineering College, Landran, for providing this opportunity to carry out the present work.

The constant guidance and encouragement received from Prof. (Dr.) Vinay Bhatia, Professor and Head, Department of Electronics and Communication Engineering, has been of great help in carrying out our present work and helped us in completing this project with success.

I would like to express a deep sense of gratitude to my Project Guide Mr. Kamlesh Singh for the guidance and support in defining the design problem and towards the completion of my project work. Without his/ her wise counsel and able guidance, it would have been impossible to complete the thesis in this manner.

We are very thankful to our college, management , training teachers and all the staff supporting us for helping us out in this project and also helping us in making this project report and other documents needed during this thesis. I am also thankful to all the faculty and staff members of Solitaire Infosys, Mohali or their intellectual support throughout the course of this work.

Submitted By:

Ramanpreet Kaur

Raj Sharma

Pallavi Bharti

Parneet Singh

Rishi Suri

Vishal Yadav

TABLE OF CONTENT

LISTS OF FIGURES

CHAPTER 1
INTRODUCTION TO PROJECT

1.1 Snake Game

This project is a game written in Java based on the game named 'Snake' which has been around since the earliest days of home computing and has re-emerged in recent years of mobile phones. This project aims to being fun and simplicity of snake game.

In the game of Snake, the player uses the arrow keys to move a "snake" around the board. As the snake finds food, it eats the food, and thereby grows larger. The game ends when the snake either moves off the screen or moves into itself. The goal is to make the snake as large as possible before that happens.

CHAPTER 2
INTRODUCTION TO JAVA

2.1 Introduction to JAVA

JAVA was developed by James Gosling at Sun Microsystems Inc in the year 1995, later acquired by Oracle Corporation. It is a simple programming language. Java makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs.

Java is a class-based, object-oriented programming language and is designed to have as few implementation dependencies as possible. A general-purpose programming language made for developers to write once run anywhere that is compiled Java code can run on all platforms that support Java. Java applications are compiled to byte code that can run on any Java Virtual Machine. The syntax of Java is similar to c/c++.

The principles for creating java were simple, robust, secured, high performance, portable, multi-threaded, interpreted, dynamic, etc. In 1995 Java was developed by James Gosling, who is known as the Father of Java. Currently, Java is used in mobile devices, internet programming, games, e-business, etc.

History of Java: James Gosling pioneered Java in June 1991 as a project called ‘Oak.’ Gosling aimed to develop a virtual machine and language with a well-known notation like C, but with more precision and simplicity than C/C++. In 1995, Java 1.0 was the first public execution. It pledged ‘Write Once, Run Anywhere’ on popular platforms with free runtimes. It was very safe and configurable with security that restricted network and file access. In a stable ‘applet’ setup, the significant web browsers soon implemented it in their standard settings.
In 1997, Sun reached out to the ISO/IEC JTC1 and then Ecma International to formalize Java, but they quickly withdrew. Java continues to be a de facto proprietary standard regulated by the Java Community Process. With the revenue generated by new vision such as the Java Enterprise Framework, Sun made several Java implementations free of charge. The critical difference is that the compiler is not present in the JRE, which differentiates between its Software Development Kit (SDK) and JRE (JRE).
On 13 November 2006, Sun launched a considerable amount of Java in the GNU General Public License as free and open-source software (GPL) (GPL). On 8 May 2007, Sun completed the process by releasing a fully accessible, all free and open-source Java’s core code, except for a small portion of the code that Sun did not copyright.

2.2 Terminologies of JAVA

Basic Terminologies of Java language is written following:

- JVM: Java Virtual Machine(JVM) is an engine that provides a runtime environment to drive the Java Code or applications. It converts Java bytecode into machine language. JVM is a part of Java Run Environment (JRE). It cannot be separately downloaded and installed. To install JVM, you need to install JRE. JVM provides a platform-independent way of executing Java source code. It has numerous libraries, tools, and frameworks. Once you run Java program, you can run on any platform and save lots of time. JVM comes with JIT(Just-in-Time) compiler that converts Java source code into low-level machine language. Hence, it runs more faster as a regular application.

- Bytecode in the Development process: As discussed, the Javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. It is saved as .class file by the compiler. To view the bytecode, a disassembler like javap can be used.
- JDK: Java Development Kit(JDK) is a software development environment used for making applets and Java applications. Java developers can use it on Windows, macOS, Solaris, and Linux. JDK helps them to code and run Java programs. It is possible to install more than one JDK version on the same computer.
- JRE: Java Runtime Environment(JRE) is a piece of a software which is designed to run other software. It contains the class libraries, loader class, and JVM. In simple terms, if you want to run Java program you need JRE. If you are not a programmer, you don't need to install JDK, but just JRE to run Java programs. Though, all JDK versions comes bundled with Java Runtime Environment, so you do not need to download and install the JRE separately in your PC.
- Garbage Collector: In Java, programmers can't delete the objects. To delete or recollect that memory JVM has a program called Garbage Collector. Garbage Collectors can recollect the objects that are not referenced. So Java makes the life of a programmer easy by handling memory management. However, programmers should be careful about their code whether they are using objects that have been used for a long time. Because Garbage cannot recover the memory of objects being referenced.
- ClassPath: The classpath is the file path where the java runtime and Java compiler look for .class files to load. By default, JDK provides many libraries. If you want to include external libraries they should be added to the classpath.

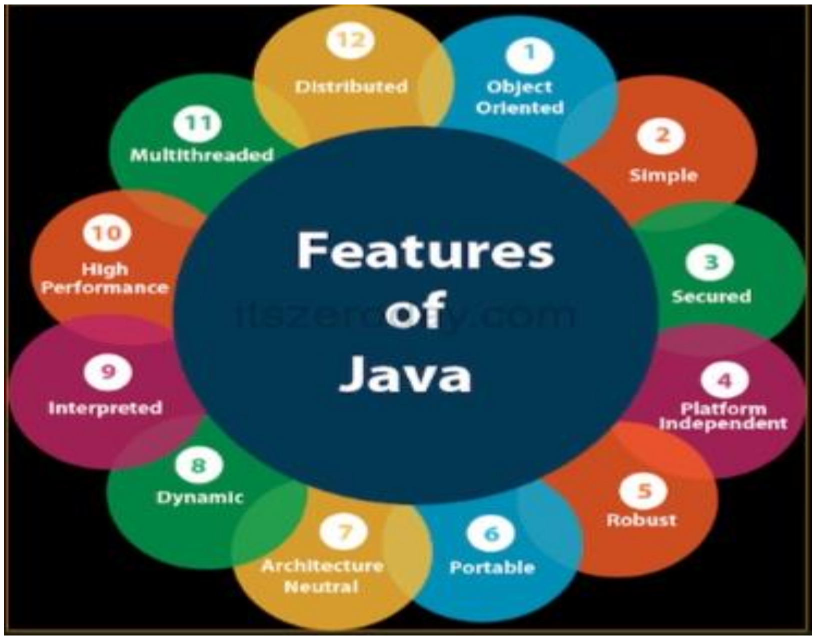
2.3 Features of Java

Following are the basic features of Java :

- Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- Platform Independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into

platform-independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

- Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.



- Robust: Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.
- Multithreaded: With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.
- Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable.

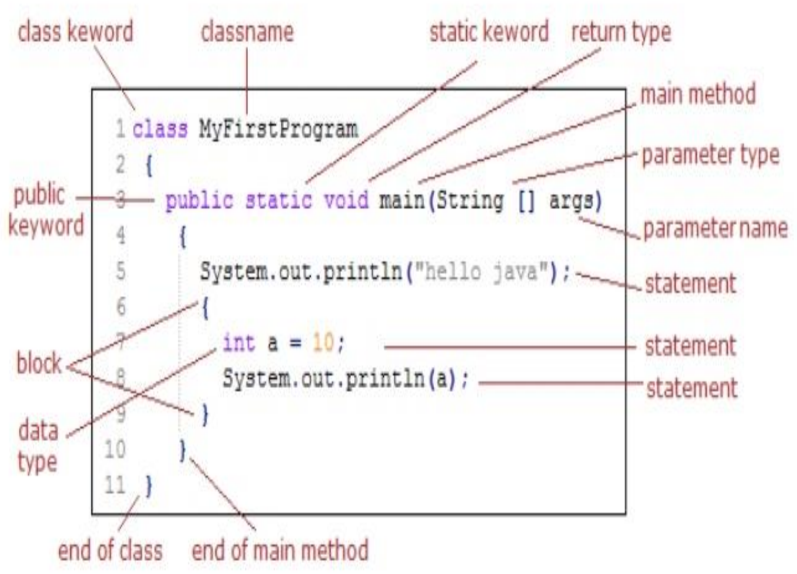
- Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

CHAPTER 3
JAVA SYNTAX

3.1 Terms used in Java program:

- Keywords - Every programming language defines a set of words which has a predefined meaning in that programming language. These words are known as keywords. You can not use these words for variable names, method names, class names or any other identifier names in your program, since these are reserved words. Java has also defined some words which has a predefined meaning in java programming language, java programmers can not use these words for naming their identifiers in a program. These reserved words are known as keywords in java. For example, in statement int a = 10, int is keyword which represents the data type of variable a. You can not use int keyword as your variable, method or class names.
- class - The class is a keyword in java which is used to define a class. In java every program must have a class. A class contains set of methods and variables. After class keyword programmers need to write the name of the class which is used to refer that class within or outside the class. In above program MyFirstProgram is the name of class, everything that is inside balanced {} after the class name are the part of class. Refer classes in java tutorial to get a complete detail about class in java.
- Statement - A statement is similar to a sentence in english language. As sentences makes a complete idea, similarly java statement makes a complete unit of execution. In above program line 5,7,8 are statements.
- Method - A method is set of statements that performs specific task or call other methods. A method has a name and a return type. The name of the method is used to refer that method within or outside the class. A class can have multiple methods. In above program main is the method name. Everything that comes in balanced { } after method name are the part of method. Every java program must have a main method if it needs to be run

independently. The main method is the starting point of execution of a program in java. Refer methods in java tutorial to get more detail about methods.



```
class MyFirstProgram
{
    public static void main(String [] args)
    {
        System.out.println("hello java");
        {
            int a = 10;
            System.out.println(a);
        }
    }
}
```

- block - A block is a group of zero or more statements. it starts with curly braces { and ends with balanced }. All statements inside balanced { } is the part of block. A block is generally used to group several statements as a single unit. A block can have another block inside it. Blocks does not have a name, they are just logical grouping of statements inside { }. Refer static and instance initializer blocks in java to get more detail about blocks in java.
- public - The keyword public is an access modifier that decides the visibility or accessibility of a member. Variables or methods declared with public keyword can be accessed outside the class. Since main method is called by JVM at the time of program execution that is why it must be declared as public, otherwise JVM won't be able to find the main method in your program and your program will not execute.
- static - The static is a keyword in java. A method or variable declared with static keyword can be called without creating the object of that class. Since JVM calls the main method without creating the object of class, that is why it must be declared as static, otherwise JVM won't be able to call the main method.
- return - The return is also a keyword in java. It is used to return value from the method to the caller of the method. Every method must have a return type, if it's not returning any

value then the return type of that method must be void. Since main method doesn't return any value, that's why it's return type is void.

- Variable and data type - Variable in java is very similar to variable in mathematics, used to store value. Data type of a variable defines what type of data that variable can store. In above program a is a variable and int is it's data type, which means a can contain only integer type of value.
- Parameter - A parameter is special kind of variable which receives the value from the caller of the method. A parameter can be used within the method in which it is declared. In above example args is a parameter of type String array. Any argument passed to program while running the program is stored in args parameter.
- System.out.println() - It is used to print the output(any string or variable) of a program on console. Anything passed to println method will be printed on console. A console is basically a window or terminal where you can pass input to program or print the output of a program. It throws the cursor to the next line after printing the desired result.
- String[] args - In java args contains the supplied command-line arguments as an array of string.

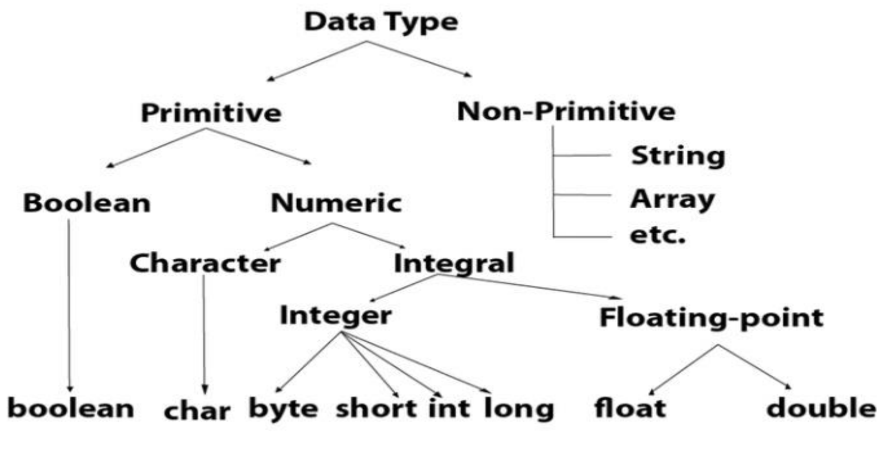
CHAPTER 4
DATATYPES AND FLOW CONTROL STATEMENTS

4.1 Datatypes in Java

Data types are different sizes and values that can be stored in the variable that is made as per convenience and circumstances to cover up all test cases. A data type is an attribute of a variable which tells the compiler or interpreter how the programmer intends to use the variable. It defines the operations that can be done on the data and what type of values can be stored.

There are two types of data types in Java:

1. Primitive Datatypes: The primitive data types include boolean, char, byte, short, int, long, float and double.
2. Non Primitive Datatypes: The non-primitive data types include Classes, Interfaces, and Arrays.



Primitive Datatypes: In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language. There are 8 types of primitive data types:

1. Boolean Datatypes: This is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions. The Boolean data type

specifies one bit of information, but its "size" can't be defined precisely. Example: Boolean one = false

2. Byte Datatypes: This is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0. The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type. Example: byte a = 10, byte b = -20
3. Short Datatypes: It is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0. The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer. Example: short s = 10000, short r = -5000
4. Int Datatype: The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0. The int data type is generally used as a default data type for integral values unless if there is no problem about memory. Example: int a = 100000, int b = -200000
5. Long Datatype: It is a 64-bit two's complement integer. Its value-range lies between - 9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int. Example: long a = 100000L, long b = -200000L
6. Float Datatype: The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F. Example: float f1 = 234.5f
7. Double Datatype: It is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The

double data type also should never be used for precise values, such as currency. Its default value is 0.0d. Example: double d1 = 12.3

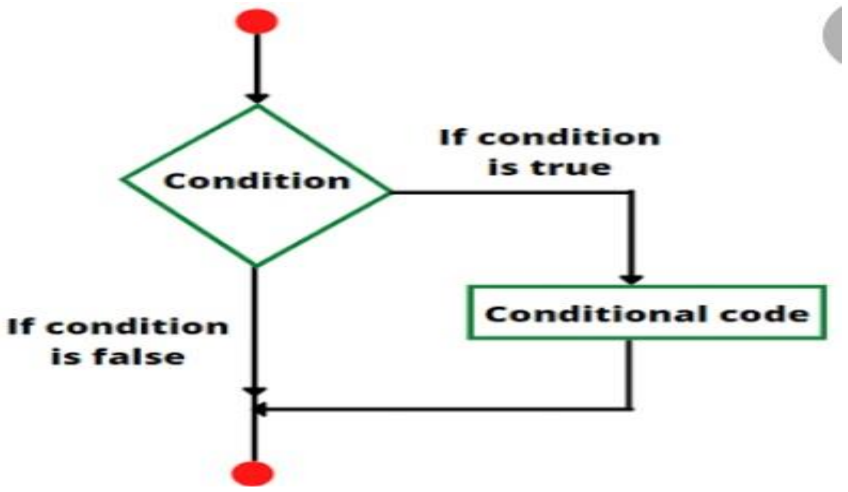
8. Char Datatype: The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters. Example: char letterA = 'A' .

Non Primitive Datatypes: Non-primitive data types are created by programmers. They are not predefined in java like primitive data types. These data types are used to store a group of values or several values. In Java programming, all non-primitive data types are simply called objects which are created by instantiating a class.

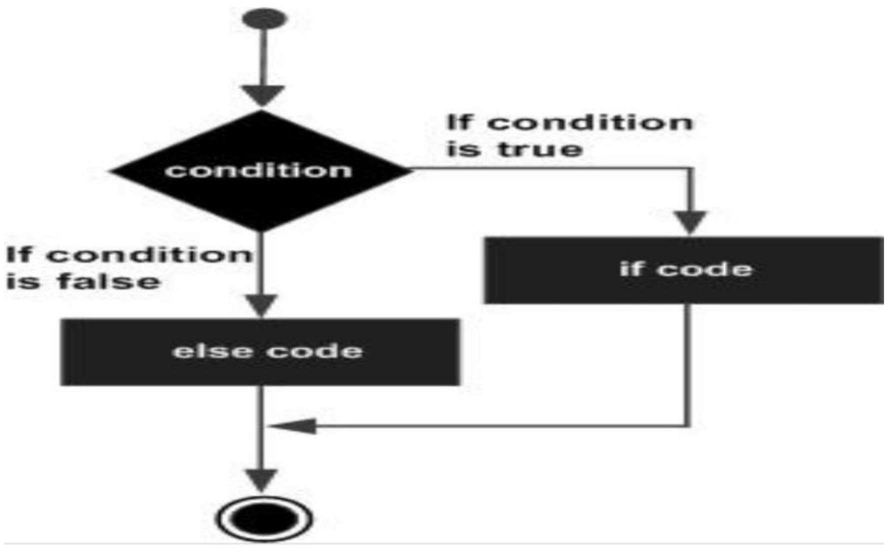
4.2 Flow Control Statements

Control Statements are used to control the execution flow of the program. Control flow or flow of control is the order in which instructions, statements, and functions call are being executed or evaluated when a program is running. The control flow statements are also called Flow Control Statements. In Java, statements inside your code are generally executed sequentially from top to bottom, in the order that they appear. There are three types of control statements:

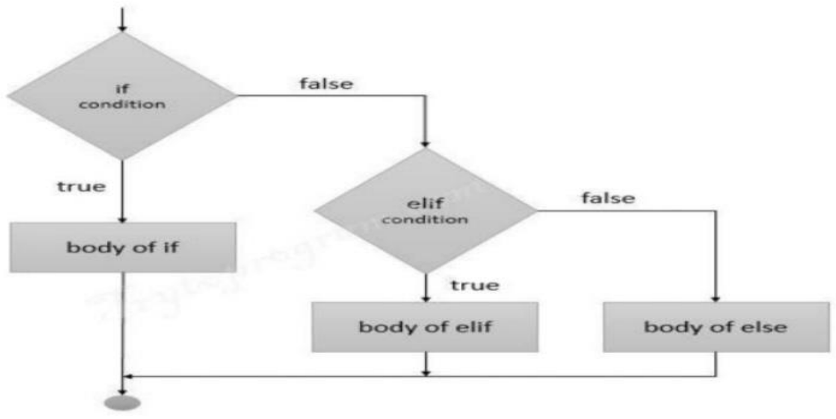
1. Decision Making statements: As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. Some decision making control statements are:
   a. Simple If statement: It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.



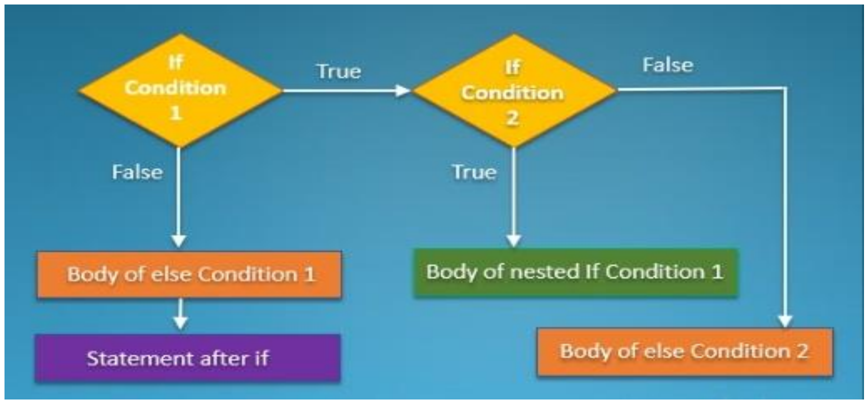
b. If-else statement: The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.



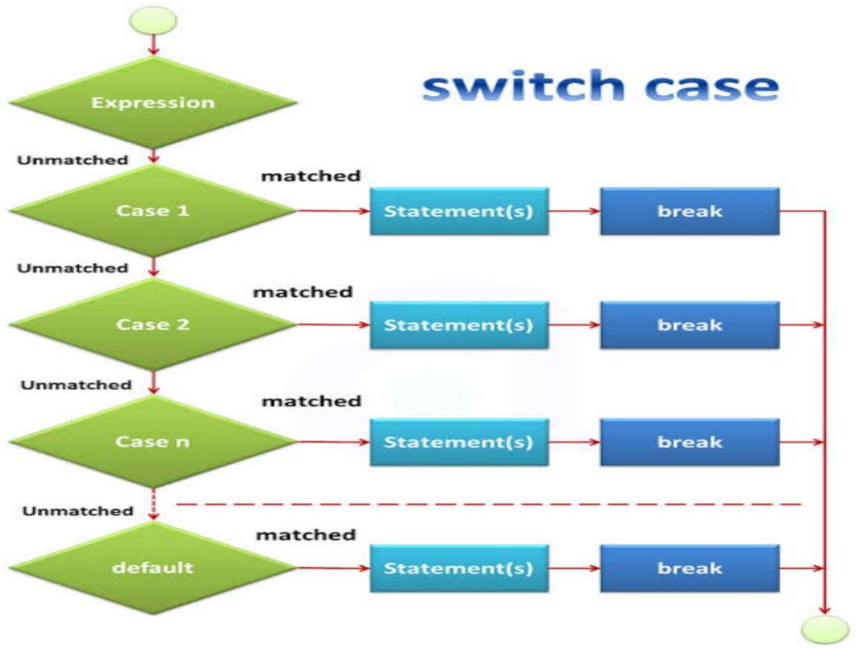
c. if-else-if statement: It contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.



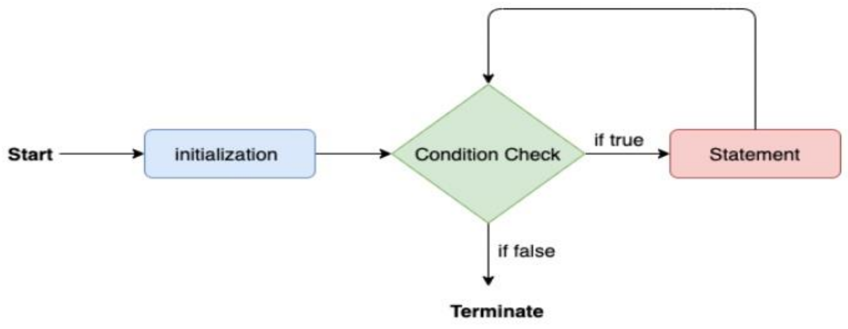
d. Nested if-statements: In this statement, the if statement can contain a if or if-else statement inside another if or else-if statement.



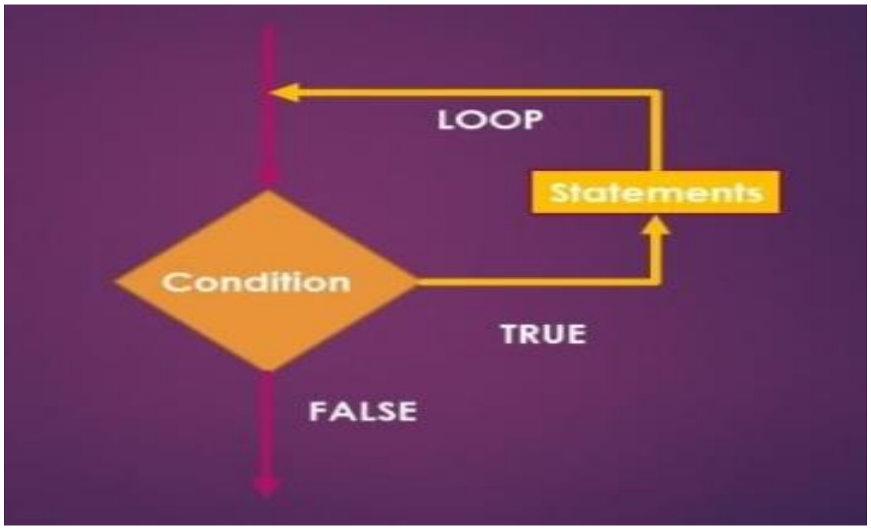
e. Switch statement: In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.



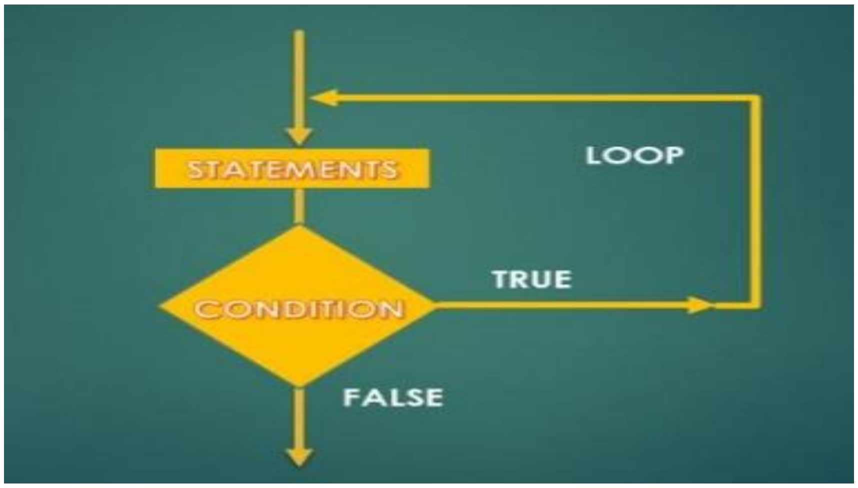
2. Loop statements: In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition. In Java, we have three types of loops that execute similarly:
   a. Java for loop: In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.



   b. Java while loop: The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop. It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

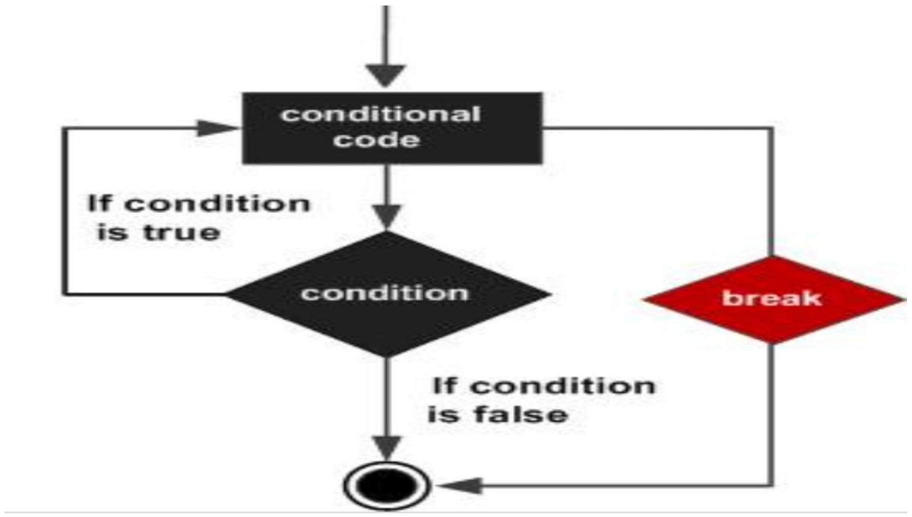


c. Java do-while loop: The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop. It is also known as the exit-controlled loop since the condition is not checked in advance.

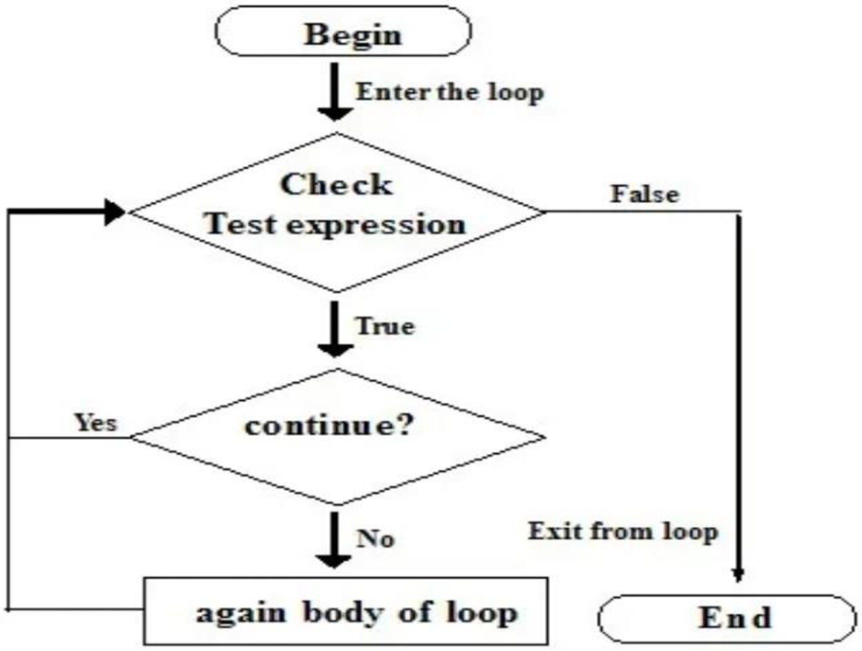


3. Jump statements: Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

a. Java break statement: It is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop. The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.



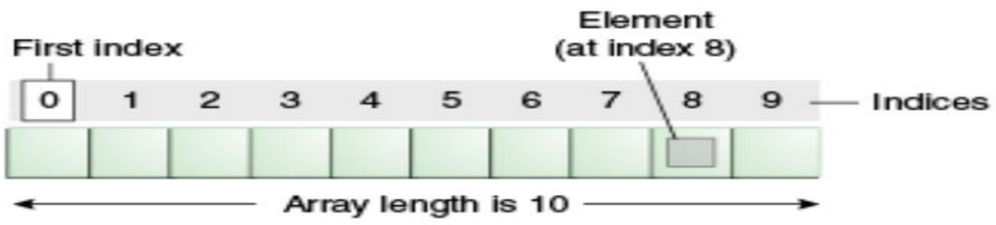
b. Java continue statement: It doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.



CHAPTER 5
ARRAY AND METHOD IN JAVA

5.1 Array in Java

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.



Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

There are two types of arrays in Java:

- Single dimensional array − A single dimensional array of Java is a normal array where, the array contains sequential elements (of same type).

--> Declaration of an Array: dataType[] arrayName;

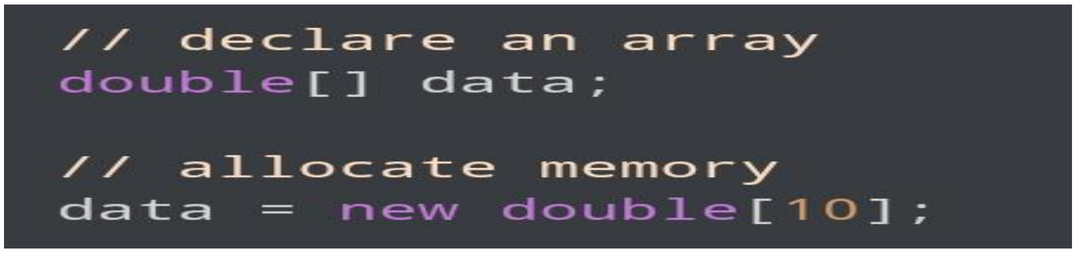
Here, dataType:- can be primitive data types like int, char, double, byte, etc. or Java objects

arrayName :- it is an identifier

For example: double[] data;

Here, 'data' is an array that can hold values of type 'double'.

● To define the number of elements that an array can hold, we have to allocate memory for the array in Java. For example:



```
// declare an array
double[] data;

// allocate memory
data = new double[10];
```

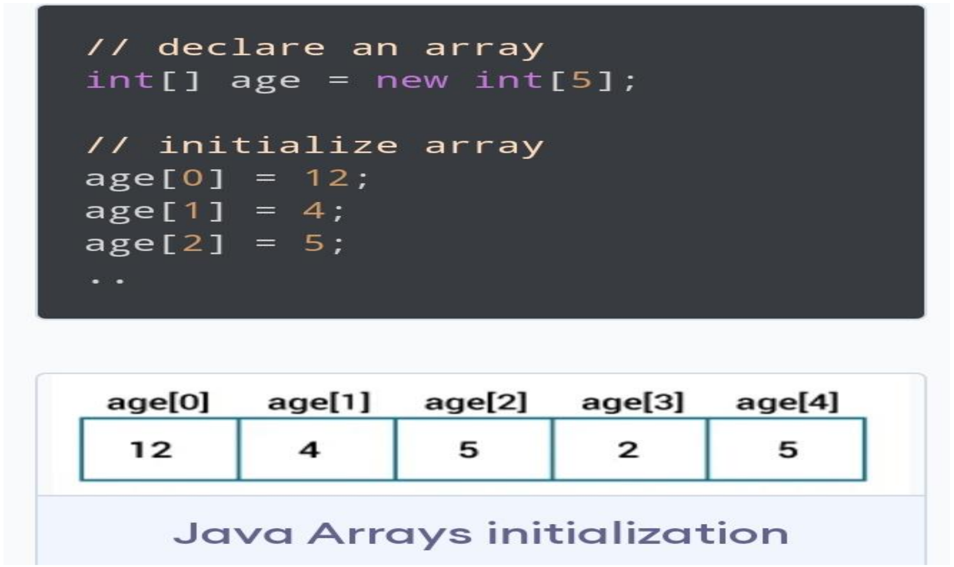
Here, the array can store 10 elements. We can also say that the size or length of the array is 10.

● In Java, we can declare and allocate the memory of an array in one single statement. For example:. double[] data = new double[10];

--> **Initialization of an Array:** In Java, we can initialize arrays during declaration. For example: int[] age = {12, 4, 5, 2, 5};

Here, we have created an array named age and initialized it with the values inside the curly brackets. we have not provided the size of the array. In this case, the Java compiler automatically specifies the size by counting the number of elements in the array (i.e. 5).

● In the Java array, each memory location is associated with a number. The number is known as an array index. We can also initialize arrays in Java, using the index number. For example:



```
// declare an array
int[] age = new int[5];

// initialize array
age[0] = 12;
age[1] = 4;
age[2] = 5;
..
```

age[0]	age[1]	age[2]	age[3]	age[4]
12	4	5	2	5

Java Arrays initialization

--> **Access Elements of an Array in Java:** We can access the element of an array using the index number. Syntax: array[index]

For example:

class Main {

  public static void main(String[] args) {

```
    int[] age = {12, 4, 5, 2, 5};
    System.out.println("Accessing Elements of Array:");     // access each array elements
    System.out.println("First Element: " + age[0]);
    System.out.println("Second Element: " + age[1]);
    System.out.println("Third Element: " + age[2]);
    System.out.println("Fourth Element: " + age[3]);
    System.out.println("Fifth Element: " + age[4]);
 }
}
```

--> **Looping Through Array Elements:** In Java, we can also loop through each element of the array. For example:

```
class Main {
 public static void main(String[] args) {
   int[] age = {12, 4, 5};     // create an array
   System.out.println("Using for Loop:");
     for(int i = 0; i < age.length; i++)         // loop through the array
       {
          System.out.println(age[i]);
       }
   }
}
```

- Multi-dimensional array − A multi-dimensional array in Java is an array of arrays. A two dimensional array is an array of one dimensional arrays and a three dimensional array is an array of two dimensional arrays.

--> Declaration and initialization of Multidimensional Array:

Syntax: data_type[1st dimension][2nd dimension][]..[Nth dimension] array_name = new data_type[size1][size2]….[sizeN];

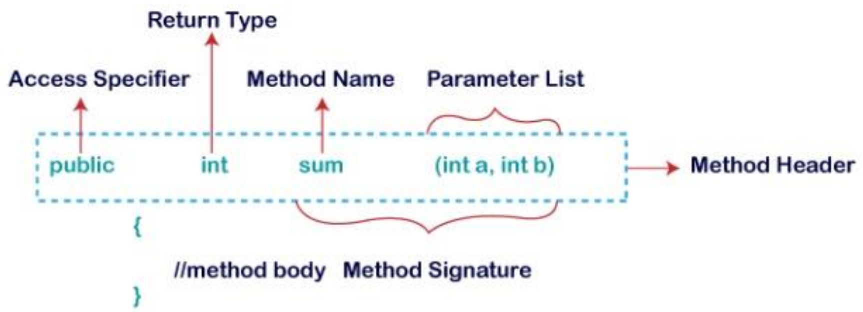
Example of 2D array:

```
class TwoDArray {
    public static void main(String[] args)
    {
        int[][] arr = { { 1, 2 }, { 3, 4 } };
        for (int i = 0; i < 2; i++)
            for (int j = 0; j < 2; j++)
                System.out.println("arr[" + i + "][" + j + "] =" + arr[i][j]);
    }
}
```

5.2 Method in Java

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the easy modification and readability of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it. The most important method in Java is the main() method. Method is also called as Function.

- Method Declaration: The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as method header.



a. Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the method name and parameter list.

b. Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method.

    i. Public: The method is accessible by all classes when we use public specifier in our application.
    ii. Private: When we use a private access specifier, the method is accessible only in the classes in which it is defined.
    iii. Protected: When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
    iv. Default: When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

  c. Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.
  d. Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be subtraction(). A method is invoked by its name.
  e. Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.
  f. Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

- Naming of Method: While defining a method, remember that the method name must be a verb and start with a lowercase letter. If the method name has more than two words, the first name must be a verb followed by adjective or noun. In the multi-word method name, the first letter of each word must be in uppercase except the first word. It is also possible that a method has the same name as another method name in the same class, it is known as method overloading.
  For Example: Single-word method name: sum(), area()
  Multi-word method name: areaOfCircle(), stringComparision()
- Types of Method: There are two types of methods in Java:

a. Predefined Methods: In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are length(), equals(), compareTo(), sqrt(), etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library. For example:

```
public class Demo   {
        public static void main(String[] args)   {
              System.out.print("The maximum number is: " + Math.max(9,7));
}
}
```

In the above example, we have used three predefined methods main(), print(), and max(). We have used these methods directly without declaration because they are predefined.

b. User defined Methods: The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement. For example:

```
public static void findEvenOdd(int num)
{      //method body
      if(num%2==0)
          System.out.println(num+" is even");
      else
          System.out.println(num+" is odd");
}
```

In above example, We have defined the above method named findevenodd(). It has a parameter num of type int.

- Call a user defined method: Once we have defined a method, it should be called. The calling of a method in a program is simple. When we call or invoke a user-defined method, the program control transfer to the called method. For example:

```
import java.util.Scanner;
```

```
public class EvenOdd  {
        public static void main (String args[])  {
              Scanner scan=new Scanner(System.in);
              System.out.print("Enter the number: ");
              int num=scan.nextInt();     //reading value from the user
              findEvenOdd(num);       //method calling
        }
//user defined method
public static void findEvenOdd(int num)  {
      if(num%2==0)     //method body
           System.out.println(num+" is even");
      else
           System.out.println(num+" is odd");
     }
}
```

CHAPTER 6

OBJECT AND CLASS IN JAVA

6.1 Object in Java

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system. An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

For example: car obj= new car; // creating an object

An object has three characteristics:

a. State: represents the data (value) of an object.
b. Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
c. Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

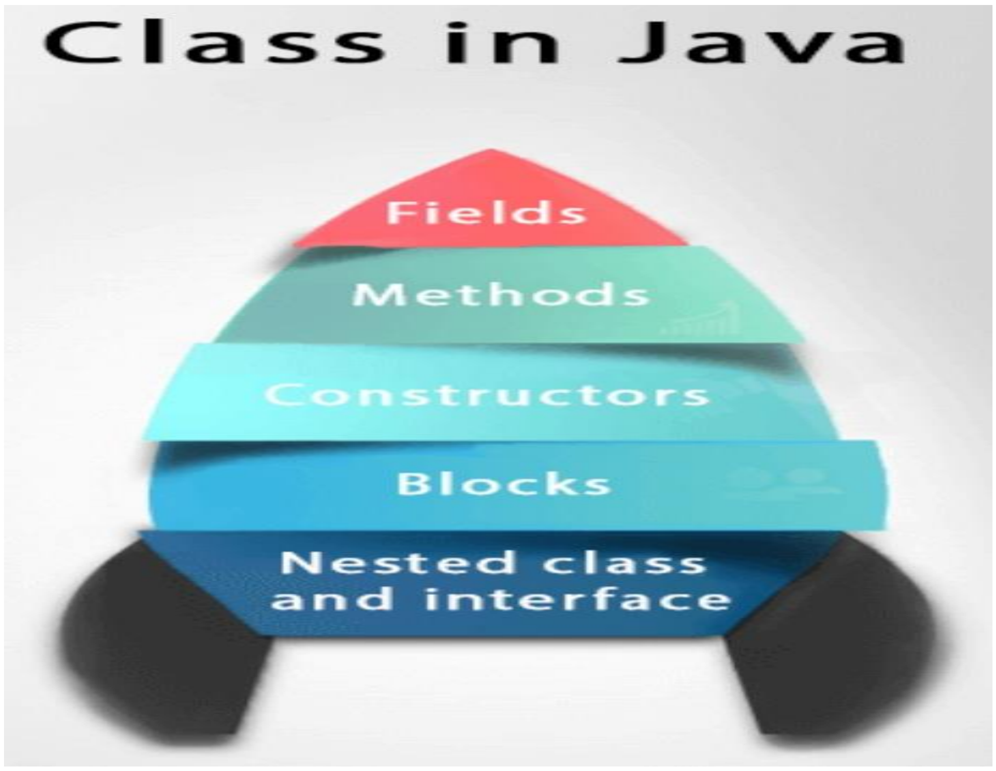
A
State
Represents the data of an object.
Behavior
represents the behavior of an object such as deposit, withdraw, etc.
B
C
Identity
It is used internally by the JVM to identify each object uniquely.

6.2 Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

Declaration of class:

```
 class <class_name>{
    field;
    method;
}
```



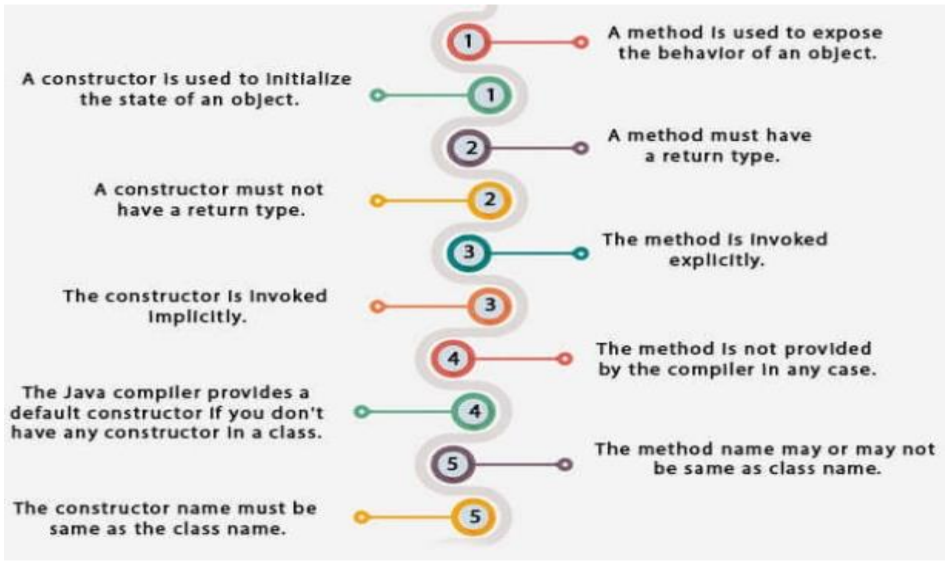
CHAPTER 7
CONSTRUCTORS IN JAVA

7.1 Constructor

Java constructors or constructors in Java is a terminology been used to construct something in our programs. A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling the constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called.

- Difference between Constructor and Method:



1 A method is used to expose the behavior of an object.
1 A constructor is used to initialize the state of an object.
2 A method must have a return type.
2 A constructor must not have a return type.
3 The method is invoked explicitly.
3 The constructor is invoked implicitly.
4 The method is not provided by the compiler in any case.
4 The Java compiler provides a default constructor if you don't have any constructor in a class.
5 The method name may or may not be same as class name.
5 The constructor name must be same as the class name.

- Need of Constructor: Think of a Box. If we talk about a box class then it will have some class variables (say length, breadth, and height). But when it comes to creating its object(i.e Box will now exist in the computer's memory), then can a box be there with no value defined for its dimensions. The answer is no.

So constructors are used to assigning values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

- Calling of Constructor: Each time an object is created using a new() keyword, at least one constructor (it could be the default constructor) is invoked to assign initial values to the data members of the same class.
- The rules for writing constructors:
    a. Constructor(s) of a class must have the same name as the class name in which it resides.
    b. A constructor in Java can not be abstract, final, static, or Synchronized.
    c. Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

7.2 Types of Constructor:

Primarily there are three types of constructors in java:

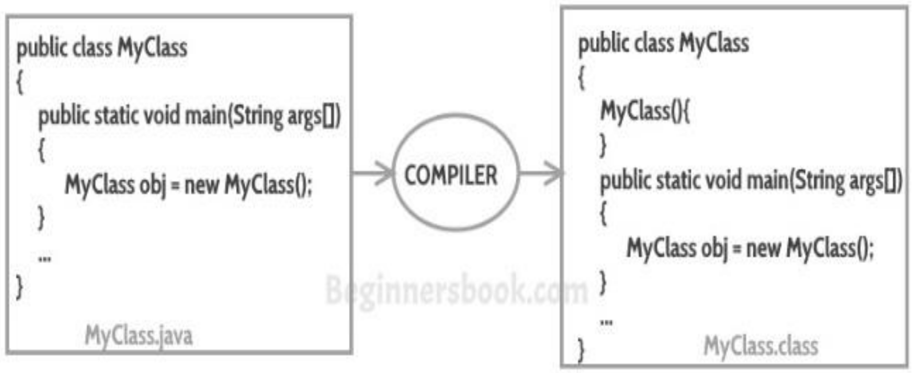
a. No-argument constructor: A constructor that has no parameter is known as the default constructor. If we don’t define a constructor in a class, then the compiler creates a default constructor(with no arguments) for the class. And if we write a constructor with arguments or no-arguments then the compiler does not create a default constructor. For Example:

```
class Demo
{
    public Demo()
    {
       System.out.println("This is a no argument constructor");
    }
    public static void main(String args[]) {
          new Demo();
    }
}
```

b. Parameterized Constructor: A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor. For Example:

```
public class Employee {
  int empId;
  String empName;
  Employee(int id, String name){
     this.empId = id;     //parameterized constructor with two parameters
     this.empName = name;
  }
  void info(){
      System.out.println("Id: "+empId+" Name: "+empName);
  }
  public static void main(String args[]){
        Employee obj1 = new Employee(10245,"Chaitanya");
        Employee obj2 = new Employee(92232,"Negan");
        obj1.info();
        obj2.info();
  }
}
```

c. Default Constructor: If you do not implement any constructor in your class, Java compiler inserts a default constructor into your code on your behalf. This constructor is known as default constructor. You would not find it in your source code(the java file) as it would be inserted into the code during compilation and exists in .class file. For Example:

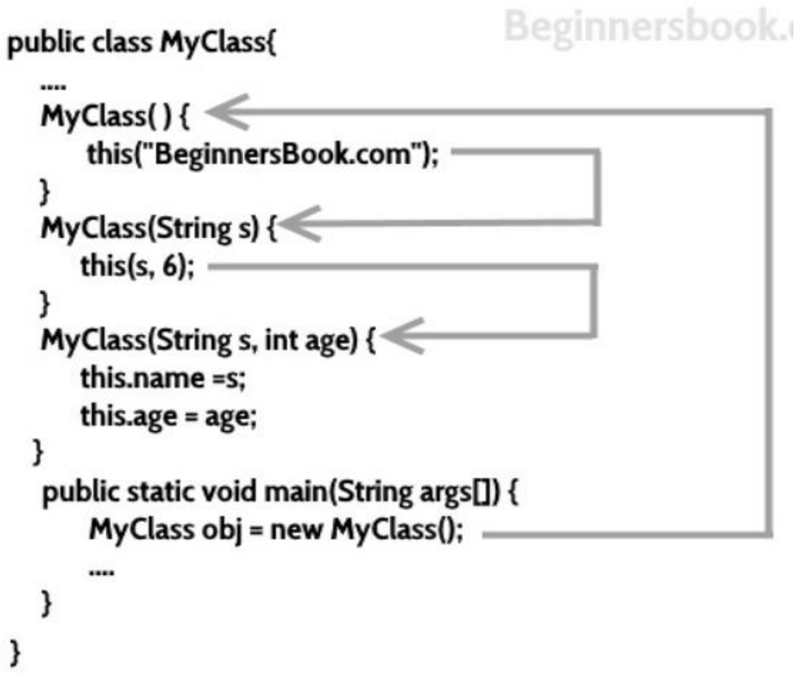


```
public class MyClass
{
   public static void main(String args[])
   {
      MyClass obj = new MyClass();
   }
   ...
}
```

```
public class MyClass
{
   MyClass(){
   }
   public static void main(String args[])
   {
      MyClass obj = new MyClass();
   }
   ...
}
```

- Constructor Chaining: Calling a constructor from the another constructor of same class is known as Constructor chaining. The real purpose of Constructor Chaining is that you can pass parameters through a bunch of different constructors, but only have the initialization done in a single place. This allows you to maintain your initializations from a single location, while providing multiple constructors to the user. If we don't chain, and two different constructors require a specific parameter, you will have to initialize that parameter twice, and when the initialization changes, you'll have to change it in every constructor, instead of just the one.

  For Example:



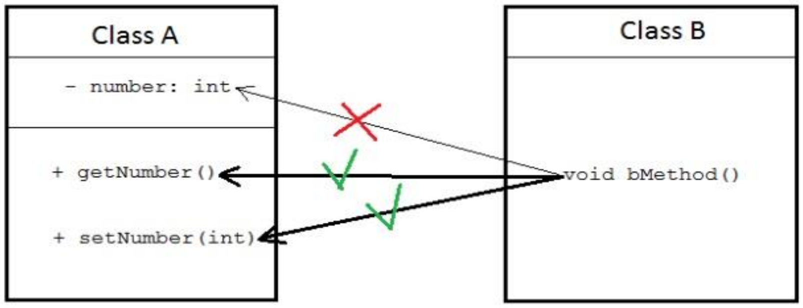
```
public class MyClass{
   ....
   MyClass( ) {
      this("BeginnersBook.com");
   }
   MyClass(String s) {
      this(s, 6);
   }
   MyClass(String s, int age) {
      this.name =s;
      this.age = age;
   }
   public static void main(String args[]) {
      MyClass obj = new MyClass();
      ....
   }
}
```

CHAPTER 8
GETTER AND SETTER METHOD IN JAVA

8.1 Introduction

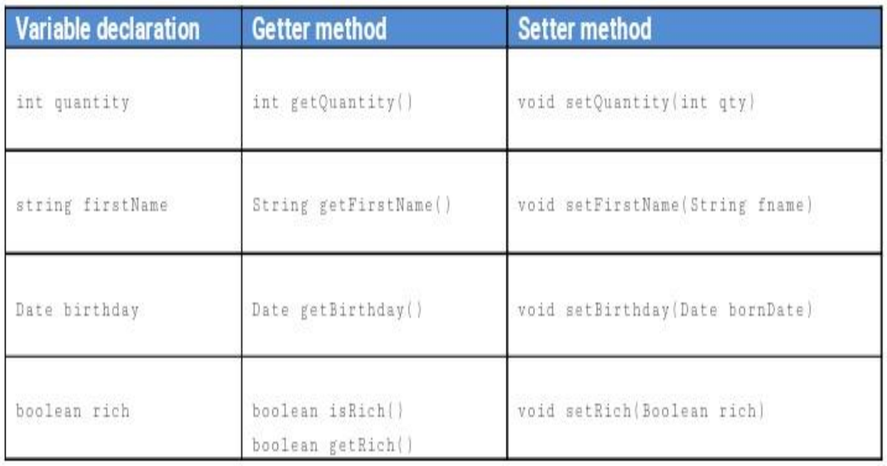
Getter and setter methods are frequently used in Java programming. Getter and setter methods in Java are widely used to access and manipulate the values of class fields. Usually, class fields are decorated with a private access specifier. Thus, to access them, public access specifiers are used with the getter and setter methods. Getter and setter are also known as accessor and mutator in Java.

- By using getter and setter, the programmer can control how his important variables are accessed and updated in a correct manner, such as changing value of a variable within a specified range.



- setter and getter methods protect a variable's value from unexpected changes by outside world - the caller code.
- When a variable is hidden by private modifier and can be accessed only through getter and setter, it is encapsulated. Encapsulation is one of the fundamental principles in object-oriented programming (OOP), thus implementing getter and setter is one of ways to enforce encapsulation in program's code.
- Naming convention for getter and setter: The naming scheme of setter and getter should follow Java bean naming convention as follows:

  getXXX() and setXXX(). where XXX is name of the variable. The following table shows some examples of getters and setters which are qualified for naming convention:



Variable declaration	Getter method	Setter method
int quantity	int getQuantity()	void setQuantity(int qty)
string firstName	String getFirstName()	void setFirstName(String fname)
Date birthday	Date getBirthday()	void setBirthday(Date bornDate)
boolean rich	boolean isRich() boolean getRich()	void setRich(Boolean rich)

- Implementing getters and setters for primitive types: With primitive types (int, float, double, boolean, char…), you can freely assign/return values directly in setter/getter because Java copies value of one primitive to another instead of copying object reference. For example:

  ```
  private float amount;
  public void setAmount(float amount) {
     this.amount = amount;
  }
  public float getAmount() {
     return this.amount;
  }
  ```

- Implementing getters and setters for common object types: String is an object type, but it is immutable which means once a String object is created, its String literal cannot be changed. In other words, every change on that String object will result in a new String object created. For example:

  ```
  private String address;
  public void setAddress(String addr) {
     this.address = addr;
  }
  public String getAddress() {
     return this.address;
  ```

CHAPTER 9

JFRAME AND JPANEL IN JAVA

9.1 JFrame

JFrame is a class of the javax.swing package that is extended by java.awt.frame. This is the top-level window, with border and a title bar. JFrame class has various methods which can be used to customize it.

JFrame is a top-level container that provides a window on the screen. A frame is actually a base window on which other components rely, namely the menu bar, panels, labels, text fields, buttons, etc. Almost every other Swing application starts with the JFrame window. Unlike a frame, JFrame has the option to hide or close the window with the help of the method setDefaultCloseOperation(int).

For any of this to work, the Java swing interface must be imported as follows:

import javax.swing.*;

- Creating a JFrame: When a new JFrame is created, you actually create an instance of the JFrame class. You can create an empty one, or one with a title. If you pass a string into the constructor, a title is created as follows:

  JFrame f = new JFrame();   // Or overload the constructor and give it a title:

  JFrame f2 = new JFrame("The Twilight Zone");

  If you run this code, however, you won't actually see an application window! The reason for this is that It has no size or location definition. You have to give it a size and make it visible to the users.

9.2 JPanel

JPanel, a part of the Java Swing package, is a container that can store a group of components. The main task of JPanel is to organize components, various layouts can be set in JPanel which provide better organization of components, however, it does not have a title bar.

- Constructors of JPanel:
  a) JPanel(): creates a new panel with a flow layout

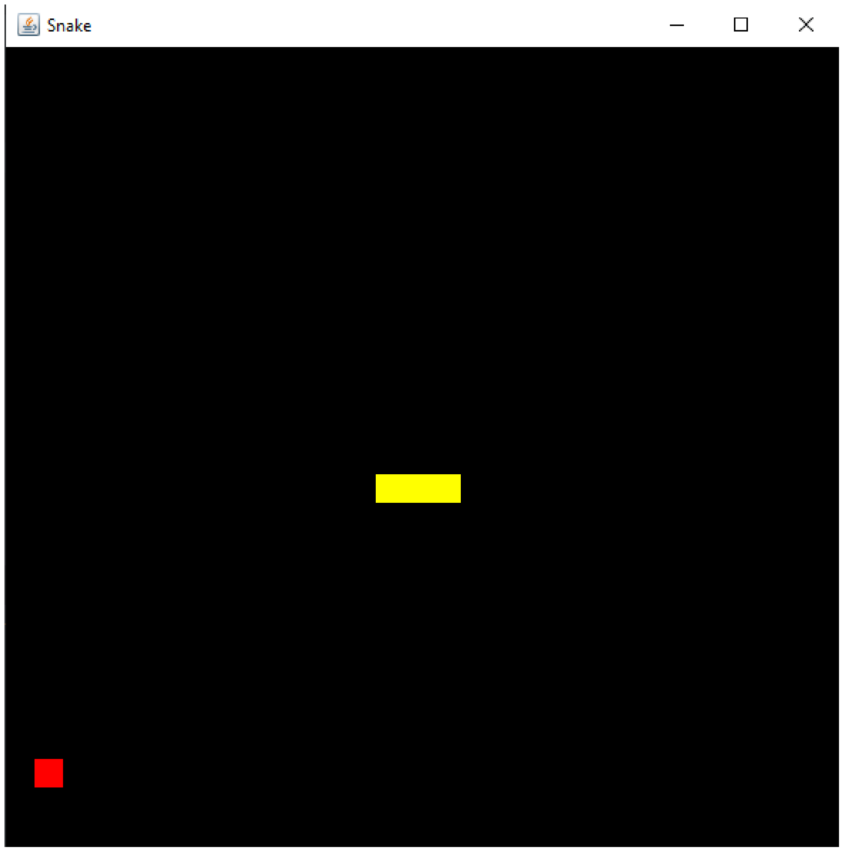
b) JPanel(LayoutManager l): creates a new JPanel with specified layoutManager
c) JPanel(boolean isDoubleBuffered): creates a new JPanel with a specified buffering strategy
d) JPanel(LayoutManager l, boolean isDoubleBuffered): creates a new JPanel with specified layoutManager and a specified buffering strategy.

- Commonly used Functions of JPanel
    a) add(Component c): Adds a component to a specified container
    b) setLayout(LayoutManager l): sets the layout of the container to the specified layout manager
    c) updateUI(): resets the UI property with a value from the current look and feel.
    d) setUI(PanelUI ui): sets the look and feel of an object that renders this component.
    e) getUI(): returns the look and feel object that renders this component.
    f) paramString(): returns a string representation of this JPanel.
    g) getUIClassID(): returns the name of the Look and feel class that renders this component.
    h) getAccessibleContext(): gets the AccessibleContext associated with this JPanel.

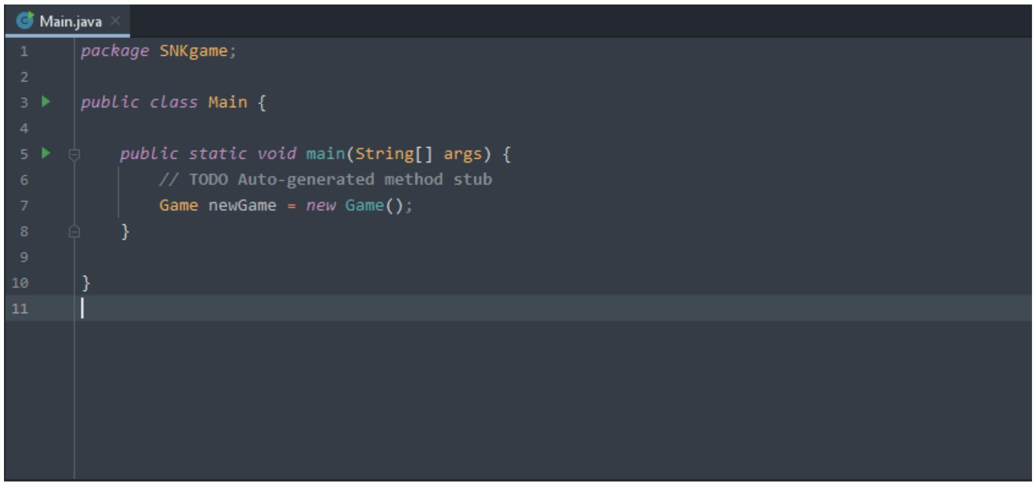
CHAPTER 10

SNAKE GAME AND CODE

10.1 Snake Game



10.2 Code:- Main File (Main.Java)



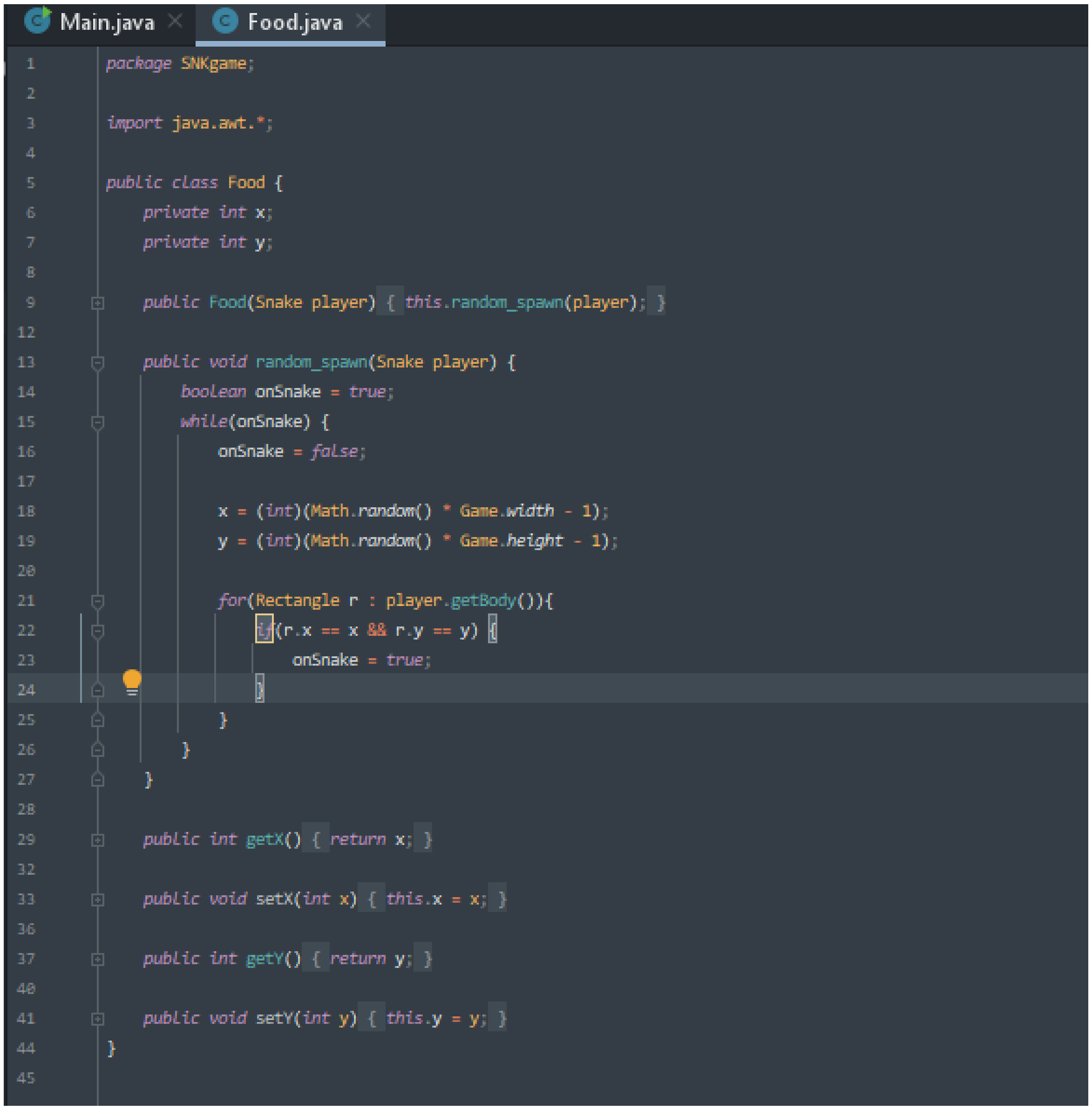
```java
package SNKgame;

public class Main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Game newGame = new Game();
    }

}
```

10.3 Food File (Food For Snake):-



```java
package SNKgame;

import java.awt.*;

public class Food {
    private int x;
    private int y;

    public Food(Snake player) { this.random_spawn(player); }

    public void random_spawn(Snake player) {
        boolean onSnake = true;
        while(onSnake) {
            onSnake = false;

            x = (int)(Math.random() * Game.width - 1);
            y = (int)(Math.random() * Game.height - 1);

            for(Rectangle r : player.getBody()){
                if(r.x == x && r.y == y) {
                    onSnake = true;
                }
            }
        }
    }

    public int getX() { return x; }

    public void setX(int x) { this.x = x; }

    public int getY() { return y; }

    public void setY(int y) { this.y = y; }
}
```

GAME FILE

(Game.Java)

```java
package SNKgame;

import javax.swing.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class Game
implements KeyListener{
   private Snake player;
   private Food food;
   private Graphics graphics;

   private JFrame window;

   public static final int width = 30;
   public static final int height = 30;
   public static final int dimension = 20;

   public Game() {
      window = new JFrame();

      player = new Snake();

      food = new Food(player);

      graphics = new Graphics(this);

      window.add(graphics);

      window.setTitle("Snake");
      window.setSize(width * dimension + 2, height * dimension + dimension + 4);
      window.setVisible(true);
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }

   public void start() {
      graphics.state = "RUNNING";
   }

   public void update() {
      if(graphics.state == "RUNNING") {
         if(check_food_collision()) {
            player.grow();
            food.random_spawn(player);
         }
         else if(check_wall_collision() || check_self_collision()) {
            graphics.state = "END";
         }
         else {
            player.move();
         }
      }
   }

   private boolean check_wall_collision() {
      if(player.getX() < 0 || player.getX() >= width * dimension
            || player.getY() < 0|| player.getY() >= height * dimension) {
         return true;
      }
      return false;
   }

   private boolean check_food_collision() {
      if(player.getX() == food.getX() * dimension && player.getY() == food.getY() * dimension) {
         return true;
      }
      return false;
   }

   private boolean check_self_collision() {
      for(int i = 1; i < player.getBody().size(); i++) {
         if(player.getX() == player.getBody().get(i).x &&
               player.getY() == player.getBody().get(i).y) {
            return true;
         }
      }
      return false;
   }

   @Override
   public void keyTyped(KeyEvent e) { }
   @Override
   public void keyPressed(KeyEvent e) {

      int keyCode = e.getKeyCode();

      if(graphics.state == "RUNNING") {
         if(keyCode == KeyEvent.VK_W && player.getMove() != "DOWN") {
            player.up();
         }

         if(keyCode == KeyEvent.VK_S && player.getMove() != "UP") {
            player.down();
         }

         if(keyCode == KeyEvent.VK_A && player.getMove() != "RIGHT") {
            player.left();
         }

         if(keyCode == KeyEvent.VK_D && player.getMove() != "LEFT") {
            player.right();
         }
      }
      else {
         this.start();
      }
   }

   @Override
   public void keyReleased(KeyEvent e) {  }

   public Snake getPlayer() {
      return player;
   }

   public void setPlayer(Snake player) {
      this.player = player;
   }

   public Food getFood() {
      return food;
   }

   public void setFood(Food food) {
      this.food = food;
   }

   public JFrame getWindow() {
      return window;
   }

   public void setWindow(JFrame window) {
      this.window = window;
   }

}
```

GRAPHICS FILE

(Graphics.java)

```
package SNKgame;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Graphics
extends JPanel
implements ActionListener{
   private Timer t = new Timer(100, this);
   public String state;

   private Snake s;
   private Food f;
   private Game game;

   public Graphics(Game g) {
      t.start();
      state = "START";

      game = g;
      s = g.getPlayer();
      f = g.getFood();

      //add a keyListner
      this.addKeyListener(g);
      this.setFocusable(true);
      this.setFocusTraversalKeysEnabled(false);
   }

   public void paintComponent(java.awt.Graphics g) {
      super.paintComponent(g);

      Graphics2D g2d = (Graphics2D) g;

      g2d.setColor(Color.black);
      g2d.fillRect(0, 0, Game.width * Game.dimension + 5, Game.height * Game.dimension + 5);

      if(state == "START") {
         g2d.setColor(Color.white);
         g2d.drawString("Press Any Key", Game.width/2 * Game.dimension - 40, Game.height / 2 *
Game.dimension - 20);
      }
      else if(state == "RUNNING") {
         g2d.setColor(Color.red);
         g2d.fillRect(f.getX() * Game.dimension, f.getY() * Game.dimension, Game.dimension,
Game.dimension);

         g2d.setColor(Color.yellow);
         for(Rectangle r : s.getBody()) {
            g2d.fill(r);
         }
      }
      else {
         g2d.setColor(Color.white);
         g2d.drawString("Your Score: " + (s.getBody().size() - 3), Game.width/2 *
Game.dimension - 40, Game.height / 2 * Game.dimension - 20);
      }
   }

   @Override
   public void actionPerformed(ActionEvent e) {
      repaint();
      game.update();
   }

}
```

SNAKE FILE

(Snake.java)

```java
package SNKgame;

import java.awt.*;
import java.util.ArrayList;

public class Snake {
   private ArrayList<Rectangle> body;
   private int w = Game.width;
   private int h = Game.height;
   private int d = Game.dimension;

   private String move; //NOTHING, UP, DOWN, LEFT, RIGHT

   public Snake() {
      body = new ArrayList<>();

      Rectangle temp = new Rectangle(Game.dimension, Game.dimension);
      temp.setLocation(Game.width / 2 * Game.dimension, Game.height / 2 *
Game.dimension);
      body.add(temp);

      temp = new Rectangle(d, d);
      temp.setLocation((w / 2 - 1) * d, (h / 2) * d);
      body.add(temp);

      temp = new Rectangle(d, d);
      temp.setLocation((w / 2 - 2) * d, (h / 2) * d);
      body.add(temp);

      move = "NOTHING";
   }

   public void move() {
      if(move != "NOTHING") {
         Rectangle first = body.get(0);

         Rectangle temp = new Rectangle(Game.dimension, Game.dimension);

         if(move == "UP") {
            temp.setLocation(first.x, first.y - Game.dimension);
         }
         else if(move == "DOWN") {
            temp.setLocation(first.x, first.y + Game.dimension);
         }
```

```
        else if(move == "LEFT") {
            temp.setLocation(first.x - Game.dimension, first.y);
        }
        else{
            temp.setLocation(first.x + Game.dimension, first.y);
        }

        body.add(0, temp);
        body.remove(body.size()-1);
    }
}

public void grow() {
    Rectangle first = body.get(0);

    Rectangle temp = new Rectangle(Game.dimension, Game.dimension);

    if(move == "UP") {
        temp.setLocation(first.x, first.y - Game.dimension);
    }
    else if(move == "DOWN") {
        temp.setLocation(first.x, first.y + Game.dimension);
    }
    else if(move == "LEFT") {
        temp.setLocation(first.x - Game.dimension, first.y);
    }
    else{
        temp.setLocation(first.x + Game.dimension, first.y);
    }

    body.add(0, temp);
}

public ArrayList<Rectangle> getBody() {
    return body;
}


public void setBody(ArrayList<Rectangle> body) {
    this.body = body;
}

public int getX() {
    return body.get(0).x;
}

public int getY() {
    return body.get(0).y ;
}

public String getMove() {
    return move;
}

public void up() {
        move = "UP";
    }
    public void down() {
        move = "DOWN";
    }
    public void left() {
        move = "LEFT";
    }
    public void right() {
        move = "RIGHT";
    }
}
```