**Assignment – 2**

**Snippet 1:**

public class Main {

public void main(String[] args) {

System.out.println("Hello, World!");

}

}

**Ans**- In this code error arises because the main method is non static.

-In java main method is entry point of the program.

-By making the main method static JVM directly invokes the main method without creating instance of class.

-This is compulsory because at starting of the class there are no objects of class

**Snippet 1 Solution -**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

**Snippet 2 –**

public class Main {

static void main(String[] args) {

System.out.println("Hello, World!");

}

}

**Error –** Main method not found in class Main

**Explanation**- this error shows because the main method is not publicly available.

-public is access modifier which makes main() method globally available.

-by making main() method publicly available JVM can identify the entry point of the program.

**Snippet 2 after correction-**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

**Snippet 3 –**

public class Main {

public static int main(String[] args) {

System.out.println("Hello, World!");

return 0;}}

**Error-** Main method must return the value of type void.

**Explanation-** main() method doesn’t return anything . its always having return type void.

-As soon as java method is terminated java program get terminate too. So there is no meaning to get return anything.

**Snippet 3 after correction –**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

return 0;

}}

**Snippet 4 –**

public class Main {

public static int main() {

System.out.println("Hello, World!");

return 0;}}

**Error –** Main method not found in class Main

**Explanation –** On compiling the code there is no error but after running it gives the error.

-String args[] is an array of String class that stores java command line arguments.

**Snippet 4 after correction–**

public class Main {

public static int main(String args[]) {

System.out.println("Hello, World!");

return 0;

}}

**Snippet 5-**

public class Main {

public static void main(String[] args) {

System.out.println("Main method with String[] args");

}

public static void main(int[] args) {

System.out.println("Overloaded main method with int[] args");

} }

**Can you have multiple main methods? What do you observe?**

**Ans-** Yes, we can have multiple main() methods but JVM always call main() method with String args[].

**Snippet 6 –**

public class Main {

public static void main(String[] args) {

int x = y + 10;

System.out.println(x);

}}

**Error-** Cannot find symbol.

**Explanation-** in this code symbol y is not declared and intialiazed.

**Snippet 6 after correction –**

public class Main {

public static void main(String[] args) {

int y=5;

int x = y + 10;

System.out.println(x); }}

**Snippet 7-**

public class Main {

public static void main(String[] args) {

int x = "Hello";

System.out.println(x);

} }

**Error-** Incompatible types.

**Explanation-** In this code we have declared variable x with int and providing String value to the x.

**Why does Java enforce type safety?**

-Error prevention

-efficient memory management

-code redability and maintainability

**Snippet 7after correction-**

public class Main {

public static void main(String[] args) {

int x = 2; //changed value of x from “hello ” to 2.

System.out.println(x);

} }

**Snippet 8 -**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!"

} }

**Error-** ')' expected

-here ); is missing.

**Explanation-** on compilation it will through error that ‘)’ expected.

**Snippet 8 after correction -**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

} }

**Snippet 9 –**

public class Main {

public static void main(String[] args) {

int class = 10;

System.out.println(class);

} }

**Error-** ‘ ; ’ expected

**Explanation-** In this snippet we used the keyword (reserved word) as variable name. So the compiler show the error.

-keywords are parts of syntax so they cannot use as identifiers.

**Snippet 9 after correction –**

public class Main {

public static void main(String[] args) {

int c = 10; //keyword class is changed to variable c.

System.out.println(c);

} }

**Snippet 10 –**

public class Main {

public void display() {

System.out.println("No parameters");

}

public void display(int num) {

System.out.println("With parameter: " + num);

}

public static void main(String[] args) {

display();

display(5);

}

}

**Error-** non-static method display(int) cannot be referenced from a static context

**Explanation –** In this snippet we cannot call the non-static methods from the main method that is static method. Because they are outside of the main class.

-for calling the display(), and display(5) we need to create the objects of the methods

**Snippet 10 after correction-**

public class Main {

public void display() {

System.out.println("No parameters");

}

public void display(int num) {

System.out.println("With parameter: " + num);

}

public static void main(String[] args) {

Main m = new Main(); //here we have created the object

m.display();

m.display(5);

} }

**Snippet 11 –**

public class Main {

public static void main(String[] args) {

int[] arr = {1, 2, 3};

System.out.println(arr[5]);

}

}

**Error –** here we get the runtime exception -> ArrayIndexOutOfBoundsException

**Explanation –** In this snipped at the initializing time we initialize with the size of 3, and we printing the int value at 5 .

We declare the small size array and printing the large size this is not allowed.

**Snippet 11 after correction–**

public class Main {

public static void main(String[] args) {

int[] arr = {1, 2, 3};

System.out.println(arr[2]);//here we print the value stored at location 2 i.e.3.

}

}

**Snippet 12-**

public class Main {

public static void main(String[] args) {

while (true) {

System.out.println("Infinite Loop");

}

}

}

**What happens when you run this code?**

When we run this code the while loop is running infinite time, Because the condition is always true.

**How can you avoid infinite loops?**

to avoid this infinite loop we need to put a condition which terminates after getting desired result. E.g. now we write the code for print values less than 5.

**Snippet 12 after correction-**

public class Main {

public static void main(String[] args) {

int n = 1;

while (n<5) { //here we terminate the while loop after n=5

System.out.println("Infinite Loop "+n);

n++;

}

}

}

**Snippet 13 –**

public class Main {

public static void main(String[] args) {

String str = null ;

System.out.println(str.length());

}

}

**Exception-** NullPointerException

**Explanation-** null is a reserved keyword in Java that signifies the absence of any value.we cannot assign null to primitives.

**Snippet 13 after correction–**

public class Main {

public static void main(String[] args) {

String str = “” ;//replace null keyword

System.out.println(str.length());

}

}

**Snippet 14 –**

public class Main {

public static void main(String[] args) {

double num = "Hello";

System.out.println(num);

}

}

**Error-** Incompatible types.

**Explanation-** In this code we have declared variable num with double and providing String value to the num.

**Why does Java enforce type safety?**

-Error prevention

-efficient memory management

-code redability and maintainability

**Snippet 14 after correction –**

public class Main {

public static void main(String[] args) {

double num = 2.6; //here we changed the value from string to double

System.out.println(num);

}

}

**Snippet 15 –**

public class Main {

public static void main(String[] args) {

int num1 = 10;

double num2 = 5.5;

int result = num1 + num2;

System.out.println(result);

}

}

**Error –** Incompatible types

**Explanation –** here num1 is int and num2 is double and we storing result into int .

-The size of double is greater than int. By changing the datatype of result we get correct output.

**Snippet 15 after correction–**

public class Main {

public static void main(String[] args) {

int num1 = 10;

double num2 = 5.5;

double result = num1 + num2; //changed datatype from int to double

System.out.println(result);

}

}

**Snippet 16 –**

public class Main {

public static void main(String[] args) {

int num = 10;

double result = num / 4;

System.out.println(result);

}

}

**What is the result of this operation?**

The output of this operation is **2.2**

**Is the output what you expected?**

Yes

**Snippet 17 –**

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 5;

int result = a \*\* b;

System.out.println(result);

}

}

**Error –** illegal start of expression

int result = a \*\* b;

**Explanation-** The problem is that \*\* is not a valid operator in Java for exponentiation.

**Snippet 17 after creation–**

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 5;

int result =(int) Math.pow(a,b);

System.out.println(result);

}

}

**Snippet 18 –**

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 5;

int result = a + b \* 2;

System.out.println(result);

}

}

**What is the output of this code?**

20

**How does operator precedence affect the result?**

Here first b gets multiplied by 2 and then addition.

**Snippet 19 –**

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 0;

int result = a / b;

System.out.println(result);

} }

**Exception-** java.lang.ArithmeticException

**Explanation-** we cannot divide int by zero which leads to exception. However, for a float or double, Java allows the operation.

**Snippet 19 after correction–**

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 1;

int result = a / b;

System.out.println(result);

} }

**Snippet 20 –**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World")

}

}

**Error-** ‘;’ expected

**How does the missing semicolon affect compilation?**

It will give the error. i.e. ‘;’ expected

**Snippet 20 after correction –**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World") ;

}

}

**Snippet 21 –**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

// Missing closing brace here

}

**What does the compiler say about mismatched braces?**

It will give the error i.e. reached end of file while parsing.

**Snippet 21 after correction –**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}// Missing closing brace here

}

**Snippet 22-**

public class Main {

public static void main(String[] args) {

static void displayMessage() {

System.out.println("Message");

}

}

}

**Error**- illegal start of expression

**Snippet 23-**

public class Confusion {

public static void main(String[] args) {

int value = 2;

switch(value) {

case 1:

System.out.println("Value is 1");

case 2:

System.out.println("Value is 2");

case 3:

System.out.println("Value is 3");

default:

System.out.println("Default case");

} } }

**Why does the default case print after "Value is 2"?**

Because there is no break statement after case 2.

**How can you prevent the program from executing the default case?**

By adding break after case 2 we prevent the program from executing the default case.

**Snippet 24 –**

class MissingBreakCase {

public static void main(String[] args) {

int level = 1;

switch(level) {

case 1:

System.out.println("Level 1");

case 2:

System.out.println("Level 2");

case 3:

System.out.println("Level 3");

default:

System.out.println("Unknown level");

}

}

}

**When level is 1, why does it print "Level 1", "Level 2", "Level 3", and "Unknown level"?**

It prints all the values because there is no break statement after any case.

**What is the role of the break statement in this situation?**

The break statement ends the execution of the closest enclosing loop or switch statement.

**Snippet 25-**

public class Switch {

public static void main(String[] args) {

double score = 85.0;

switch(score) {

case 100:

System.out.println("Perfect score!");

break;

case 85:

System.out.println("Great job!");

break;

default:

System.out.println("Keep trying!");

}

}

}

**Why does this code not compile?**

Because double is not allowed in switch case.

**What does the error tell you about the types allowed in switch expressions?**

Error tells that constant label of type int is not compatible with switch selector type double. The value of the expressions in a switch-case statement must be an ordinal type i.e. integer, char, short, long, etc. Float and double are not allowed.

**How can you modify the code to make it work?**

By changing the datatype of score it will work.

**Snippet 25 after correction -**

public class Switch {

public static void main(String[] args) {

int score = 85;

switch(score) {

case 100:

System.out.println("Perfect score!");

break;

case 85:

System.out.println("Great job!");

break;

default:

System.out.println("Keep trying!");

}

}

}

**Snippet 26 –**

public class Switch {

public static void main(String[] args) {

int number = 5;

switch(number) {

case 5:

System.out.println("Number is 5");

break;

case 5:

System.out.println("This is another case 5");

break;

default:

System.out.println("This is the default case");

}

}

}

**Why does the compiler complain about duplicate case labels?**

A compiler complain about duplicate case labels because it's an implementation restriction.

**What happens when you have two identical case labels in the same switch block?**

If two cases in a switch statement have the same label, the second case will never be executed.