Section 1

Snippet 1:

```
public class Main
    {
       public void main(String[] args)
       {
          System.out.println("Hello, World!");
       }
 }
```

What error do you get when running this code?

- Can’t find main method-missing static

Snippet 2:

```
public class Main
{
   static void main(String[] args)
  {
      System.out.println("Hello, World!");
  }
}
```

What happens when you compile and run this code?

- Error is occured
- Error- Main method not found in class main, please define the main method as:
  public static void main(String[] args)

-

Snippet 3:

```
public class Main {
public static int main(String[] args)
 {
    System.out.println("Hello, World!");
     return 0;
} }
```

What error do you encounter? Why is void used in the main method?

- Main method must return a value of type void in class main, please
  define the main method as:
  public static void main(String[] args)
- Void is used because the main method is not knowing to return any values.

Snippet 4:

```
public class Main {
public static void main() {
System.out.println("Hello, World!");
} }
```

What happens when you compile and run this code? Why is String[] args needed?

- Main method not found in class main
- Entry Point for the JVM – It looks for main method with signature.
  public static void main(String args[])

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?

-Yes, The code will compile successfully. The Java Virtual Machine (JVM) does not recognize the public static void main(int[] args) method as the entry point of the program.

Snippet 6:

```
public class Main {
public static void main(String[] args) {
int x = y + 10; System.out.println(x); } }
```

What error occurs? Why must variables be declared?

-Symbol – y is not found

-Variable is declared, memory is allocated for it according to its type

Snippet 7:

```
public class Main {
public static void main(String[] args) {
int x = "Hello";
System.out.println(x); }
}
```

What compilation error do you see? Why does Java enforce type safety?

-string cannot converted into int

-Java enforces type safety to prevent errors, ensure reliable code.

Snippet 8:

```
public class Main {
public static void main(String[] args) {
System.out.println("Hello, World!" }
}
```

What syntax errors are present? How do they affect compilation?

- ‘)’ or ‘;’ expected
- syntax is incomplete

Snippet 9:

```
public class Main {
public static void main(String[] args) {
 int class = 10; System.out.println(class); }
}
```

What error occurs? Why can't reserved keywords be used as identifiers?

- Class is a reserved keyword in java. Reserved keywords are special words that are part of the Java language syntax.
- It would create ambiguity and conflicts.

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); }
 }
```

What happens when you compile and run this code? Is method overloading allowed?

-Code is not compile successfully. It will encounter error because non-static methods cannot be called directly from a static context without creating an instance of the class.

- Yes. In this code, you have successfully overloaded the display method.

Snippet 11:

```
public class Main {
 public static void main(String[] args) {
int[] arr = {1, 2, 3}; System.out.println(arr[5]); }
}
```

What runtime exception do you encounter? Why does it occur?

- ArrayIndex OutOfBound exeception is occurred.
- Because In that array there are only 3 elements. We cannot access arr[5].

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? How can you avoid infinite loops?

- It will continuously print "Infinite Loop" to the console without stopping, because the condition in the while loop is true, so the loop never terminates.
- Use a counter that increments or decrements with each iteration, and set a condition for the loop to terminate when the counter reaches a specific value.

Snippet 13:

```
public class Main {
 public static void main(String[] args) {
 String str = null;
System.out.println(str.length()); }
}
```

What exception is thrown? Why does it occur?

- NullPointerException is thrown.
- the code attempts to call str.length(), but since str is null, there's no String object to determine the length of, causing the exception.

Snippet 14:

```
 public class Main {
public static void main(String[] args) {
double num = "Hello";
System.out.println(num); }
}
```

What compilation error occurs? Why does Java enforce data type constraints?

- error- incompatible type- String cannot be converted to double.
- Java is a statically typed language, means types are checked at compile time. Reducing the runtime errors and bugs.

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); }
}
```

What error occurs when compiling this code? How should you handle different data types in Operations

- conversion from double to int is not allow in java
- By using typecasting.

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? Is the output what you expected?
- output 2.0
- expected 2.5

Because num/4 is int so output 2 and stored in double so it display 2.0

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); }
}
```

What compilation error occurs? Why is the ** operator not valid in Java?•

- Illegal start of expression
- Because java does not recognized ** as an operator.

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? How does operator precedence affect the result?

- output is 20
- In this expression a + b * 2, the multiplication (*) operator has higher precedence than the addition (+) operator. Therefore, the multiplication is performed first, followed by the 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); }
}
```

What runtime exception is thrown? Why does division by zero cause an issue in Java?

- ArithmeticException is thrown
- Mathematically, division by zero is undefined because there's no number that can multiply with zero to give a non-zero number.

Snippet 20:

```
 public class Main {
public static void main(String[] args) {
System.out.println("Hello, World") }
}
```

What syntax error occurs? How does the missing semicolon affect compilation?

- ; expected
- Because each statement in java must end with ; to signify end of statement

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?

- The error message indicates that the compiler reached the end of the file (end of file) while still expecting a closing brace to match an earlier opening brace.

Snippet 22:

```
public class Main {
public static void main(String[] args) {
static void displayMessage() {
System.out.println("Message"); }
}
 }
```

What syntax error occurs? Can a method be declared inside another method?

- The static keyword and method declarations are not allowed inside other methods
- No, can't declared

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"); }
}
}
```

Error to Investigate: Why does the default case print after "Value is 2"? How can you prevent the program from executing the default case?

- Because there are no break statements after each case block.
- To prevent the program from executing the default case after a case matches, you need to use the break statement at the end of each case block.

Snippet 24:

```
public 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"); }
}
}
```

Error to Investigate: When level is 1, why does it print "Level 1", "Level 2", "Level 3", and "Unknown level"? What is the role of the break statement in this situation?

- Because there is no break statement.
- The break statement is used to exit from the switch block immediately after the execution of a case block.

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!"); }
}
}
```

Error to Investigate: Why does this code not compile? What does the error tell you about the types allowed in switch expressions? How can you modify the code to make it work?

- double are not allowed in switch case.
- To make the code compile, you need to use a type that is allowed in a switch expression. In this case, you can use an int type instead of double

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"); }
 }
 }
```

Error to Investigate: Why does the compiler complain about duplicate case labels? What happens when you have two identical case labels in the same switch block?

- The compiler complain about duplicate case labels because each case label within a switch statement must be unique.
- If there are two identical case label the compiler cannot determined which block of code to execute leading to a conflict.