1. In Object-oriented programming(OOP), classes and objects have attributes. **Attributes** are data stored inside a class or instance and represent the state or quality of the class or instance. In short, attributes store information about the instance. Also, attributes should not be confused with class functions also known as methods. One can think of attributes as noun or adjective, while methods are the verb of the class.

i) Class Attributes
ii) Instance Attributes

2. **Compile Time Polymorphism:** Whenever an object is bound with their functionality at the compile-time, this is known as the compile-time polymorphism. At compile-time, java knows which method to call by checking the method signatures. So this is called compile-time polymorphism or static or early binding. Compile-time polymorphism is achieved through method overloading.

The following is an example where compile-time polymorphism can be observed.

```
// Java program to demonstrate
// compile-time polymorphism
public class GFG {

    // First addition function
    public static int add(int a, int b)
    {
        return a + b;
    }

    // Second addition function
    public static double add(
        double a, double b)
    {
        return a + b;
    }
    // Driver code
    public static void main(String args[])
    {
        // Here, the first addition
        // function is called
        System.out.println(add(2, 3));

        // Here, the second addition
        // function is called
        System.out.println(add(2.0, 3.0));
    }
}
Output:
5
5.0
```

3. Abstraction is the concept of object-oriented programming that "shows" only essential attributes and "hides" unnecessary information. The main purpose of abstraction is hiding the unnecessary details from the users. Abstraction is selecting data from a larger pool to show only relevant details of the object to the user. It helps in reducing programming complexity and efforts. It is one of the most important concepts of OOPs.

Example of Abstraction:

```
#include <iostream>

using namespace std;


class Summation {

  private:

    // private variables

    int myNum1, myNum2, myNum3

  public:
    void sum(int inNum1, int inNum2)

    {

        myNum1 = inNum1;

        myNum2 = inNum2;

        myNum3 = myNum1 + myNum2;

        cout << "Sum of the two number is : " << myNum3< <endl;

    }

};

int main()

{

    Summation mySum;

    mySum.sum(5, 4);

    return 0;

}
```

In this case the variables myNum1, myNum2 and myNum3 are private, thereby in accessible to any code other than the class Summation. In this example the variables are set to values passed in as arguments to the sum method. This is not a very true example - often the values would NOT be set just before being used like this, but it shows the reality of the implementation.

Output:

```
Sum of the two number is: 9
```

4.

The following example demonstrates the use of the dynamic_cast operator:

```
#include <iostream>
using namespace std;

struct A {
  virtual void f() { cout << "Class A" << endl; }
};
```

```
struct B : A {
  virtual void f() { cout << "Class B" << endl; }
};

struct C : A {
  virtual void f() { cout << "Class C" << endl; }
};

void f(A* arg) {
  B* bp = dynamic_cast<B*>(arg);
  C* cp = dynamic_cast<C*>(arg);

  if (bp)
    bp->f();
  else if (cp)
    cp->f();
  else
    arg->f();
};

int main() {
  A aobj;
  C cobj;
  A* ap = &cobj;
  A* ap2 = &aobj;
  f(ap);
  f(ap2);
}
```

See the output of the above example:

```
Class C
Class A
```

The function f() determines whether the pointer arg points to an object of type A, B, or C. The function does this by trying to convert arg to a pointer of type B, then to a pointer of type C, with the dynamic_cast operator. If the dynamic_cast operator succeeds, it returns a pointer that points to the object denoted by arg. If dynamic_cast fails, it returns 0.

You may perform downcasts with the dynamic_cast operator only on polymorphic classes. In the above example, all the classes are polymorphic because class A has a virtual function. The dynamic_cast operator uses the runtime type information generated from polymorphic classes.

5.

Exception handling in C++ consists of three keywords: try, throw and catch:

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The throw keyword throws an exception when a problem is detected, which lets us create a custom error.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

6. Functions for file handling

1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file

7

Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

8. **EXE** is a **file** extension for an executable **file** format. An executable is a **file** that contains a program — that is, a particular kind of **file** that is capable of being executed or run as a program in the computer. To make a executable file, we will need a separate tool for it called Launch4j.

9.

Definition: An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an *exception object*, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called *throwing an exception*.

10.

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

1. Extending the Thread class
2. Implementing the Runnable Interface