1. Java objects' state and behavior
-> In Java, instances of a class are known as objects. Every object has state and behavior in the form of instance fields and methods respectively.

```java
public class Person {
  // state of an object
  int age;
  String name;

  // behavior of an object
  public void set_value() {
    age = 20;
    name = "Robin";
  }
  public void get_value() {
    System.out.println("Age is " + age);
    System.out.println("Name is " + name);
  }

  // main method
  public static void main(String [] args) {
    // creates a new Person object
    Person p = new Person();

    // changes state through behavior
    p.set_value();
  }
}
```

2. Java instance
-> Java instances are objects that are based on classes. For example, Bob may be an instance of the class Person.

-> Every instance has access to its own set of variables which are known as instance fields, which are variables declared within the scope of the instance. Values for instance fields are assigned within the constructor method.

```java
public class Person {
  int age;
  String name;

  // Constructor method
  public Person(int age, String name) {
    this.age = age;
    this.name = name;
  }

  public static void main(String[] args) {
    Person Bob = new Person(31, "Bob");
    Person Alice = new Person(27, "Alice");
  }
}
```

3. Java dot notation
-> In Java programming language, we use . to access the variables and methods of an object or a Class.

-> This is known as dot notation and the structure looks like this-

```
instanceOrClassName.fieldOrMethodName

public class Person {
  int age;

  public static void main(String [] args) {
    Person p = new Person();

    // here we use dot notation to set age
    p.age = 20;

    // here we use dot notation to access age and print
    System.out.println("Age is " + p.age);
    // Output: Age is 20
  }
}
```

4. Constructor Method in Java
-> Java classes contain a constructor method which is used to create instances of the class.

-> The constructor is named after the class. If no constructor is defined, a default empty constructor is used.

```
public class Maths {
  public Maths() {
    System.out.println("I am constructor");
  }
  public static void main(String [] args) {
    System.out.println("I am main");
    Maths obj1 = new Maths();
  }
}
```

5. Creating a new Class instance in Java
-> In Java, we use the new keyword followed by a call to the class constructor in order to create a new instance of a class.

-> The constructor can be used to provide initial values to instance fields.

```
public class Person {
  int age;
  // Constructor:
  public Person(int a) {
    age = a;
  }

  public static void main(String [] args) {
    // Here, we create a new instance of the Person class:
    Person p = new Person(20);
    System.out.println("Age is " + p.age); // Prints: Age is 20
  }
}
```

6. Reference Data Types
-> A variable with a reference data type has a value that references the memory address of an instance. During variable declaration, the class name is used as the variable's type.

```java
public class Cat {
  public Cat() {
    // instructions for creating a Cat instance
  }

  public static void main(String[] args) {
    // garfield is declared with reference data type `Cat`
    Cat garfield = new Cat();
    System.out.println(garfield); // Prints: Cat@76ed5528
  }
}
```

7. Constructor Signatures
-> A class can contain multiple constructors as long as they have different
parameter values. A signature helps the compiler differentiate between the
different constructors.

-> A signature is made up of the constructor's name and a list of its parameters.

```java
// The signature is `Cat(String furLength, boolean hasClaws)`.
public class Cat {
  String furType;
  boolean containsClaws;

  public Cat(String furLength, boolean hasClaws) {
    furType = furLength;
    containsClaws = hasClaws;
  }
  public static void main(String[] args) {
    Cat garfield = new Cat("Long-hair", true);
  }
}
```

8. null Values
-> null is a special value that denotes that an object has a void reference.

```java
public class Bear {
  String species;
  public Bear(String speciesOfBear;) {
    species = speciesOfBear;
  }

  public static void main(String[] args) {
    Bear baloo = new Bear("Sloth bear");
    System.out.println(baloo); // Prints: Bear@4517d9a3
    // set object to null
    baloo = null;
    System.out.println(baloo); // Prints: null
  }
}
```

9. The body of a Java method
-> In Java, we use curly brackets {} to enclose the body of a method.

-> The statements written inside the {} are executed when a method is called.

```java
public class Maths {
  public static void sum(int a, int b) { // Start of sum
    int result = a + b;
```

```
      System.out.println("Sum is " + result);
    } // End of sum


    public static void main(String [] args) {
      // Here, we call the sum method
      sum(10, 20);
      // Output: Sum is 30
    }
}
```

10. Method parameters in Java
-> In java, parameters are declared in a method definition. The parameters act as
variables inside the method and hold the value that was passed in. They can be used
inside a method for printing or calculation purposes.

-> In the example, a and b are two parameters which, when the method is called,
hold the value 10 and 20 respectively.

```
public class Maths {
  public int sum(int a, int b) {
    int k = a + b;
    return k;
  }

  public static void main(String [] args) {
    Maths m = new Maths();
    int result = m.sum(10, 20);
    System.out.println("sum is " + result);
    // prints - sum is 30
  }
}
```

11. Java Variables Inside a Method
-> Java variables defined inside a method cannot be used outside the scope of that
method.

//For example, `i` and `j` variables are available in the `main` method only:

```
public class Maths {
  public static void main(String [] args) {
    int i, j;
    System.out.println("These two variables are available in main method only");
  }
}
```

12. Returning info from a Java method
-> A Java method can return any value that can be saved in a variable. The value
returned must match with the return type specified in the method signature.

The value is returned using the return keyword.

```
public class Maths {

  // return type is int
  public int sum(int a, int b) {
    int k;
    k = a + b;
```

```
    // sum is returned using the return keyword
    return k;
  }

  public static void main(String [] args) {
    Maths m = new Maths();
    int result;
    result = m.sum(10, 20);
    System.out.println("Sum is " + result);
    // Output: Sum is 30
  }
}
```

13. Declaring a Method
-> Method declarations should define the following method information: scope (private or public), return type, method name, and any parameters it receives.

```
// Here is a public method named sum whose return type is int and has two int parameters a and b
public int sum(int a, int b) {
  return(a + b);
}
```