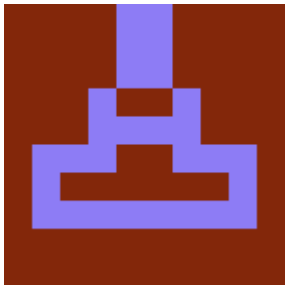
Classes



+5

Published Aug 4, 2021•**Updated Jan 2, 2023**

Contribute to Docs

In Java, **classes** are blueprints or templates for objects in Java. They detail the general structure and data for an object including information such as properties, attributes, and method behavior. Classes are classified as a reference data type.

Syntax

```
accessModifier class ClassName {
  dataType attributeOne;
  dataType attributeTwo;
  dataType attributeN;

  static void classMethod {
    // Method code here
  }
}
```

- Class names must always be in "PascalCase" and match the name of the file (e.g. ClassName.java).
- Java uses the class keyword for creating classes.
- They use an accessModifier (public, private, and protected) to determine its visibility to other files.
- Inside the class "blueprint" are members.

Class Instances

In Java, instances are objects that are based on existing classes.

Every instance has access to its own set of variables known as instance fields. These are variables declared within the scope of the instance and supplied with new values within the class constructor method during initialization.

For example, Bob and Alice may each be defined as instances of the class called Person with the new keyword:

```
// Person.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");

    System.out.println(Bob.name + " is " + Bob.age + ".");
    System.out.println(Alice.name + " is " + Alice.age + ".");
  }
}
```

Each instance of the `Person` class has an `age` and `name` field. When initialized, they are passed as arguments into the class constructor. The example from above would return the following output:

```
Bob is 31.
Alice is 27.
```

Abstract Classes

Classes can also use the `abstract` keyword to supply common method implementations to multiple subclasses. Any class that contains abstraction (methods, fields, etc.), then it must also be abstract:

```
// Person.java
abstract class Person {
  int age;
  String name;

  abstract void talk(String message);
}
```