JDK - Java Development Kit

Java Runtime Environment (JRE) + Development Tools (Debugger,Compiler,JavaDoc)

JRE - Java Runtime Environment

JVM + set of libraries/loader/other supporting files... like math, swingetc, util, lang, awt, and runtime libraries.

1. to run Java program
2. JRE is a piece of a software which is designed to run other software

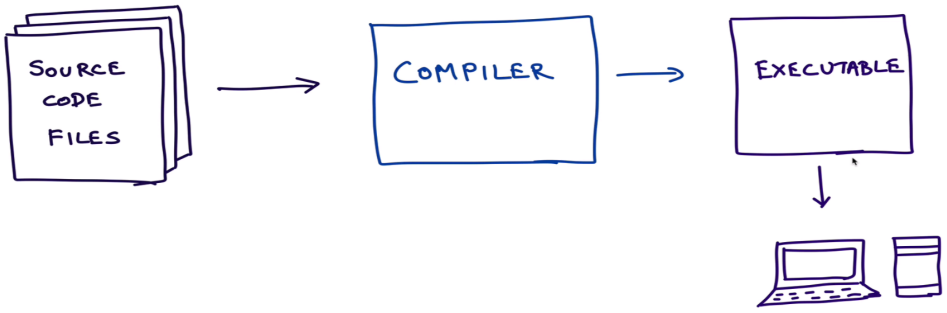
JVM - Java Virtual Machine

1. executes Java byte code and provides an environment for executing it.
2. It is a specification that provides a runtime environment in which Java bytecode can be executed.
3. JVM is responsible for executing the java program line by line hence it is also known as an interpreter.
4. It can also run those programs which are written in any language and compiled to Java bytecode.
5. JVM comes with a JIT(Just-in-Time) compiler that converts Java source code into low-level machine language. Hence, it runs faster as a regular application.

JDK is platform dependent, JRE is also platform dependent, but JVM is platform independent.

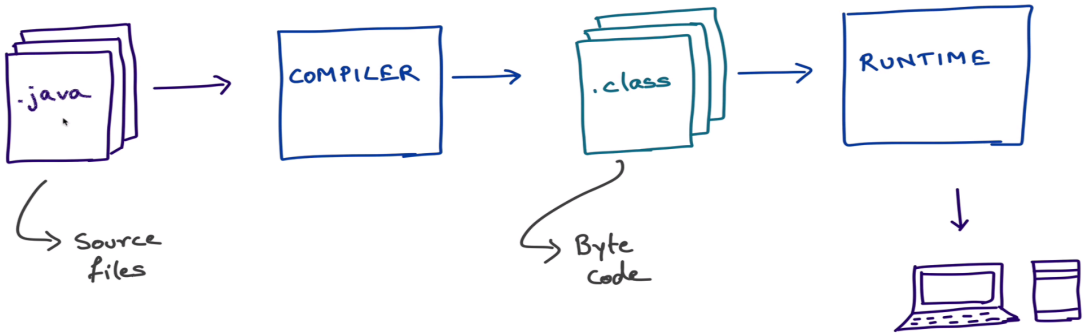
javabrains.io

Predecessor - C / C++



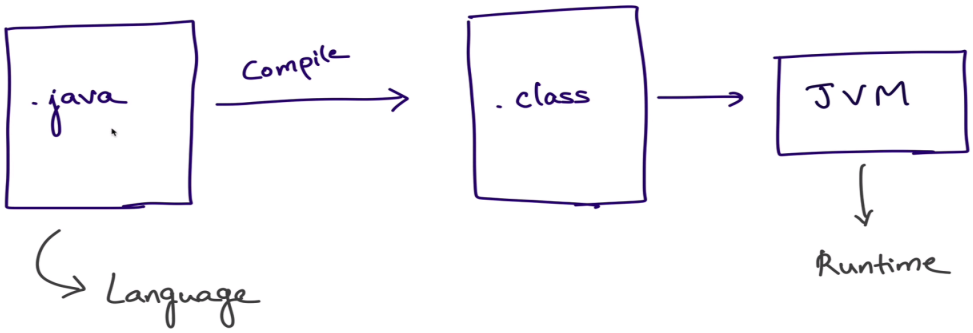
javabrains.io

The Java model



javabrains.io

Language vs runtime



javabrains.io

Java Development Kit

- Java SDK
- Contains compiler and other tools to build Java applications
- Not necessary to run Java programs
- Comes with runtime

javabrains.io

Java Runtime Environment

- Required to run Java applications
- Runs byte code on the machine it is on
- Language agnostic
- Contains standalone JVM

javabrains.io

Java Virtual Machine

- Runs on a computer any program that's compiled to byte code
- It acts like an abstract "virtual" computer
- Has byte code loader, verifier and interpreter
- Can convert byte code to machine specific code

javabrains.io

Things to remember

- Java implements call by value
- Any value passed as parameters is “copied” to the method arguments
- If the argument is an object reference, the reference is copied
- The reference copy points to the same instance as the source object reference

javabrains.io

Interview Question

Does Java support scope closures?

javabrains.io

Encapsulation revisited

```
public class Car {

    private int seats;

    public void run() {
        // Can access seats here
    }
}
```

```
public class SportsCar extends Car {

    @Override
    public void run() {
        // Cannot access seats here.
    }

}
```

javabrains.io

Encapsulation revisited

```
public class Car {

    private int seats;

    public void run() {
        // Can access seats here
    }
}
```

```
public class SportsCar extends Car {

    public void run() {
        // Can seats be accessed here?
    }

}
```

javabrains.io

Inheritance

```
class A {
    public String m;
}
```

```
class B extends A {
    public int m;
}
```

"Hides" the m in super class

javabrains.io

Inheritance

```
class A {
    public String m;
}
```

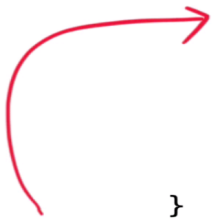
```
class B extends A {
    public int m;

    public void printValue() {
        this.m = 10;
        System.out.println(m);
    }

    public void printSuperClassValue() {
        super.m = "Hello";
        System.out.println(super.m);
    }
}
```

javabrains.io

Method overriding

```
class A {
    public String findName() {

    }
}
```

```
class B extends A {

    private String findName() {

    }


}
```



attempting to assign weaker access privileges

javabrains.io

Interfaces vs Abstract classes

Define "Contract"

"Template" or Starting point for classes

javabrains.io

Prevent inheritance

```
final class A {

}

class B extends A {
}
```

Cannot inherit from final 'A'

javabrains.io

Prevent method overrides

```
class Rabbit extends Pet {

    final public void move() {

    }
}

class WildRabbit extends Rabbit {

    public void move() {

    }
}
```

'move()' cannot override 'move()' in 'Rabbit'; overridden method is final

javabrains.io

Default methods

```
interface Drivable {

    public default void drive() {
        System.out.println("Driving");
    }

}
```

```
class SportCar implements Drivable {


}
```

javabrains.io

Default methods

```
interface Drivable {
    public default void drive() {
        System.out.println("Driving");
    }
}

interface FuelVehicle {
    public default void drive() {
        System.out.println("Driving with fuel");
    }

}
```

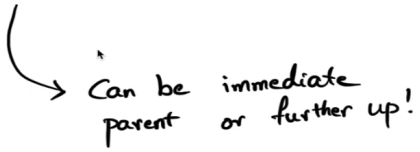
```
class SportCar implements Drivable, FuelVehicle {


}
```

SportCar inherits unrelated defaults for drive() from types Drivable and FuelVehicle

javabrains.io

Polymorphism

```
SuperClass reference = subclassInstance;
```



javabrains.io

Casting with Object references

```
public interface Drivable {
    public void drive();
}
```

```
public interface FuelVehicle {
   public void refuel (Fuel f);
}
```

```
Drivable car = new SportCar();
car.drive();
FuelVehicle vehicle = (FuelVehicle) car;
vehicle.refuel(new Fuel());
```