



Basics of Java



Table of Contents



- ▶ History of Java
- ▶ Java Specification



1

History of Java



History of Java

- ▶ Java is a **general-purpose programming language**
- ▶ That is **class-based, object-oriented**, and designed to have **as few dependencies as possible**
- ▶ It is intended to **Write Once, Run Anywhere (WORA)**
- ▶ Applications are **compiled** to **bytecode** that can run on any **Java Virtual Machine (JVM)**



► History of Java

- **Sun Microsystems** released the first public implementation as **Java 1.0 in 1996**
- Major web browsers incorporated **Java applets** and Java became popular
- **As of 2006**, Sun released much of its Java Virtual Machine (JVM) as **free and open-source software (FOSS)**, under the terms of the **GNU General Public License (GPL)**.



► History of Java

- Following **Oracle Corporation's acquisition** of Sun Microsystems in 2009–10, Oracle has described itself as the **steward of Java technology**.
- Java software runs on everything **from laptops to data centers, game consoles to scientific supercomputers**.



2

Java Specification



▶ Java Specification

▶ What is JVM? :

- ▶ JVM is a **virtual machine**
- ▶ It provides a **runtime environment** for Java **bytecode**
- ▶ It also **runs** programs in **other languages** compiled to Java bytecode
- ▶ **JVM, JRE, and JDK** are **platform dependent** because the configuration of each OS is different.

▶ Java Specification



▶ What is JVM? :

- ▶ However, **Java is platform-independent**
- ▶ The JVM performs the following **main tasks**:
 - ▶ **Loads** code
 - ▶ **Verifies** code
 - ▶ **Executes** code
 - ▶ **Provides** runtime **environment**

▶ Java Specification



▶ What is JRE? :

- ▶ Java Runtime Environment is a **software package**
- ▶ It **bundles the libraries** (jars), the **Java Virtual Machine** and other components
- ▶ To execute any Java application, **you need JRE** installed
- ▶ JREs can be downloaded as **part of JDKs** or **separately**



Java Specification

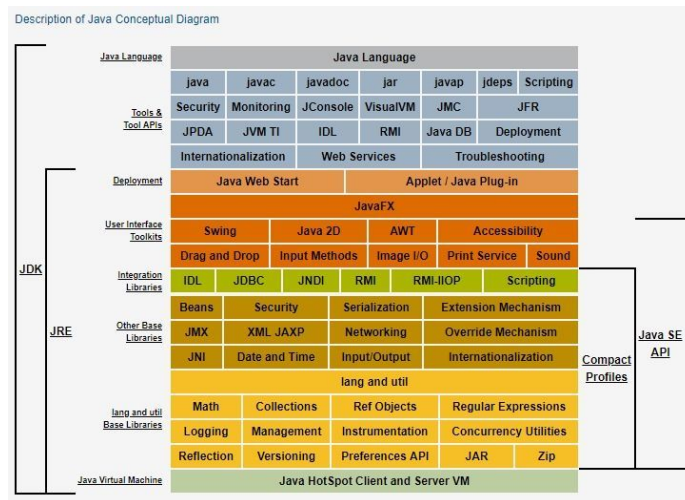
► What is JDK? :

- Java Development Kit is a **superset of JRE**
- It contains everything that **JRE has** along with **development tools for developing, debugging, and monitoring**
- You need JDK **when** you need to **develop** Java applications



Java Specification

► Java Conceptual Diagram :





A Simple Java Program



Table of Contents



- ▶ A Simple Java Program
- ▶ Create, Compile and Run



1

A Simple Java Program

A Simple Java Program



► Welcome Message from Java :

```
1 public class Welcome {  
2     public static void main(String[] args) {  
3         // Display message 'Welcome to Java!' on the console  
4         System.out.println("Welcome to Java!");  
5     }  
6 }  
7
```

```
1 Welcome to Java!  
2
```




▶ A Simple Java Program

▶ Welcome Message from Java :

- ▶ Line 1 defines **a class**
- ▶ Every Java program must have **at least one** class
- ▶ Each class has a name

```
1 public class Welcome {  
2     public static void main(String[] args) {  
3         // Display message 'Welcome to Java!' on  
4         System.out.println("Welcome to Java!");  
5     }  
6 }  
7
```

```
1 Welcome to Java!  
2
```



▶ A Simple Java Program

▶ Welcome Message from Java :

- ▶ Line 2 defines **the main method**
- ▶ Program **starts from the main** method

```
1 public class Welcome {  
2     public static void main(String[] args) {  
3         // Display message 'Welcome to Java!' on  
4         System.out.println("Welcome to Java!");  
5     }  
6 }  
7
```

```
1 Welcome to Java!  
2
```



▶ A Simple Java Program

▶ Welcome Message from Java :

- ▶ Line 3 is **a comment** →
- ▶ Java comments are preceded by two slashes (//) on a line,
- ▶ Or enclosed between /* and */ for several lines

```
1 public class Welcome {  
2     public static void main(String[] args) {  
3         // Display message 'Welcome to Java!' on  
4         System.out.println("Welcome to Java!");  
5     }  
6 }  
7
```

```
1 Welcome to Java!  
2
```



▶ A Simple Java Program

▶ Welcome Message from Java :

- ▶ Line 4 is **a statement** →
"System.out.println"
- ▶ It displays the string
Welcome to Java!
- ▶ Every Java statement **ends**
with a semicolon (;)

```
1 public class Welcome {  
2     public static void main(String[] args) {  
3         // Display message 'Welcome to Java!' on  
4         System.out.println("Welcome to Java!");  
5     }  
6 }  
7
```

```
1 Welcome to Java!  
2
```



▶ A Simple Java Program

▶ Welcome Message from Java :

- ▶ Line 5 and 6 **terminates** two **code blocks** that group the program's components

```
1 public class Welcome {  
2     public static void main(String[] args) {  
3         // Display message 'Welcome to Java!' on  
4         System.out.println("Welcome to Java!");  
5     }  
6 }  
7
```

- ▶ In Java, **each block begins with** an opening brace '{' and **ends with** a closing brace '}'

```
1 Welcome to Java!  
2
```



2

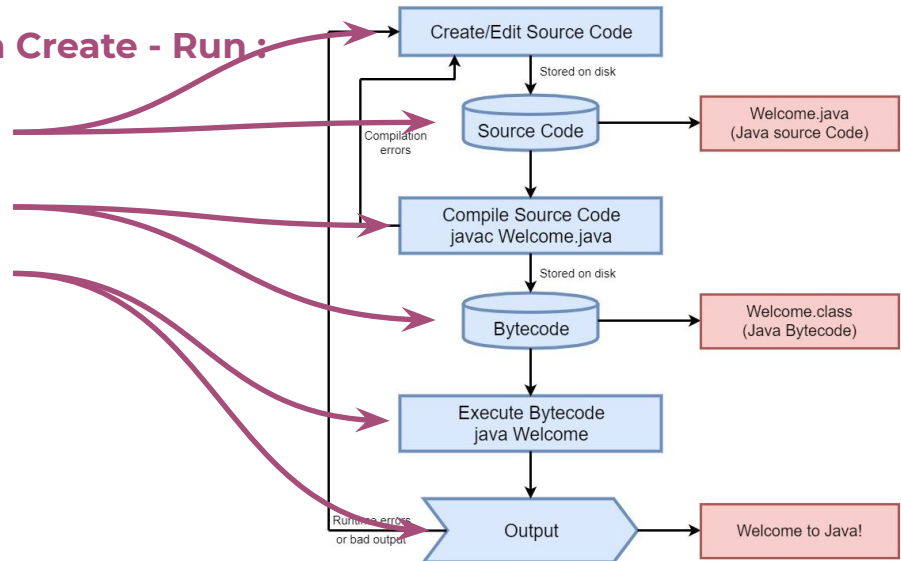
Create, Compile and Run



Create, Compile and Run

Steps through Create - Run :

- **Create**
- **Compile**
- **Run**



Running and Building Locally



Table of Contents



- ▶ **What is Building and Compiling?**
- ▶ **Building JAR Files**



1 What is Building and Compiling?

► What is Building and Compiling? »

► **Compiling :**

- ▷ Compiling is the process of converting **source code** files **into standalone software artifact(s)**
- ▷ These artifacts are **executable files**

► What is Building and Compiling? »

► **Building :**

- ▷ Building is a **broader concept**
- ▷ It consists of :
 - ▷ **Generating** sources (sometimes)
 - ▷ **Compiling** sources
 - ▷ **Compiling test sources**
 - ▷ **Executing tests** (unit tests, integration tests, etc)
 - ▷ **Packaging** (into jar, war, ejb-jar, ear)
 - ▷ **Generating reports**



2

Building JAR Files



▶ Building JAR Files

- ▶ JAR stands for **Java Archive**
- ▶ It is a kind of **zip file**
- ▶ It is a **platform-independent** file (As long as the platform has at least JVM)
- ▶ It holds :
 - ▷ All application content like :
 - ▷ **Class files**
 - ▷ **Resources** (images, sound files, Manifest file (optional))



Building JAR Files

- ▶ Compilation with **"javac App.java"**
- ▶ It gives **".class"** file
- ▶ **"java App"** runs
- ▶ **"jar -cvfe App.jar App App.class"** gives JAR
- ▶ **"java -jar App.jar"** runs the JAR file

```
MyMac:Desktop home$ cd JavaApp/  
MyMac:JavaApp home$ ls  
App.java  
MyMac:JavaApp home$ javac App.java  
MyMac:JavaApp home$ ls  
App.class App.java  
MyMac:JavaApp home$ java App  
hello world!  
MyMac:JavaApp home$ jar -cvfe App.jar App App.class  
added manifest  
adding: App.class(in = 412) (out= 286)(deflated 30%)  
MyMac:JavaApp home$ ls  
App.class App.jar App.java  
MyMac:JavaApp home$ java -jar App.jar  
hello world!  
MyMac:JavaApp home$
```



Building JAR Files

- ▶ **"jar -cvfe App.jar App App.class"**
 - create**
 - verbose**
 - file (jar file)**
 - executable**
 - JAR File Name**
 - Main Class**
 - Java File**



THANKS!

Any questions?

