

PRG37(8)1

# Programming in Java

# PRG 37(8)1

IMPORTANT INFORMATION

## DURATION

2

weeks

## ASSESSMENTS

1 Project [M1 & M2]

1 Tests



# Outcomes

---

- An understanding of integrated knowledge of programming techniques and concepts as contested to construct computing systems using tools and services to develop computing systems that consider platform constraints, supports version control, tracks requirements and bugs, and automates building.
- The ability to identify, analyse, evaluate, critically reflect on and address complex problems, applying evidence-based solutions and theory-driven arguments through the use of application programming interfaces and frameworks when implementing solutions.
- An understanding of a range of methods to construct multi-tiered applications, evaluate and verbalize the value of using the different levels of logic separation.
- The ability to develop and communicate a solid understanding of the more advanced concepts of programming inclusive of data structures, design patterns and RESTful API implementation.
- The ability to take full responsibility for their own work, decision-making and use of resources to solve problems in unfamiliar and variable contexts exposed by different technologies and methodologies for tasks and be able to judge the relative merits of these to choose between the alternatives.

# Java Programming

- Java is a popular programming language, created in 1995.
- It is owned by Oracle, and more than 3 billion devices run Java.
- It is used for:
  - ✓ Mobile applications (specially Android apps)
  - ✓ Desktop applications
  - ✓ Web applications
  - ✓ Web servers and application servers
  - ✓ Games
  - ✓ Database connection
- Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

# Concepts of Java

---

- Key Core Java concepts:

- 1) **Java Fundamentals** - the main variant of Core Java, typically used for developing regular desktop applications.
- 2) **Collections**: involves managing mainly data structures, like lists, maps, sets, etc., which provide different algorithms to handle storage.
- 3) **OOPs Concepts**: Classes, Objects, Encapsulation, Polymorphism, Inheritance.
- 4) **Packages**: form the Core Java concepts that allow the organisation and grouping of related classes and interfaces.

# Concepts of Java

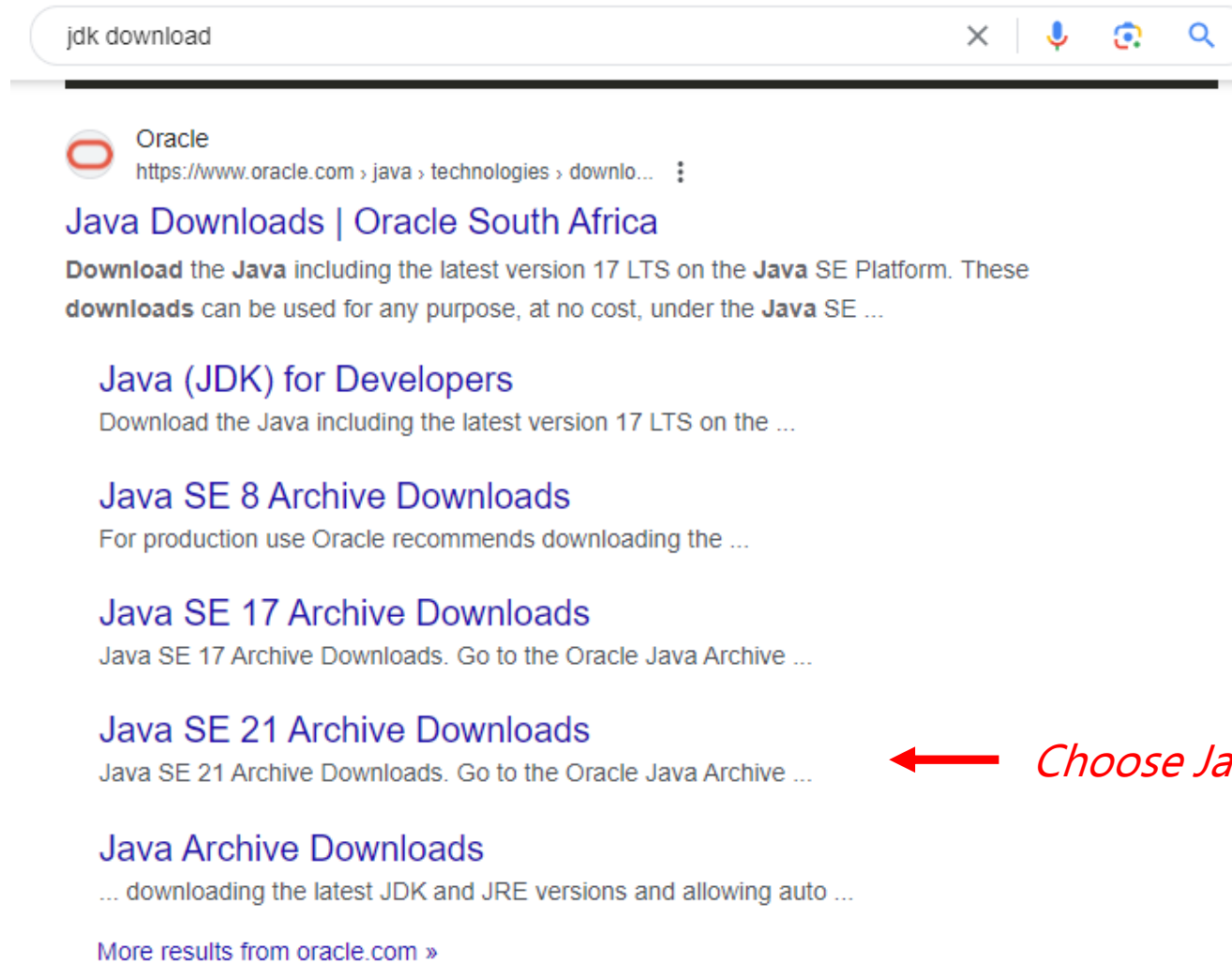
---

- Key GUI and Server-side Java concepts:

- 5) **Java Swing:** a Graphical User Interface in the concept of Core Java that helps to develop desktop apps with a quality graphical interface.
- 6) **Java Database Connectivity (JDBC):** a fundamental technology that allows Java applications to interact with databases for CRUD operations.
- 7) **PostgreSQL (Postgres):** a powerful open-source relational database management system (RDBMS) known for its reliability, scalability, and extensibility.
- 8) **Java Servlets:** server-side programs written in Java that handle requests and generate responses in a web environment.

# Installing Java

- Search:



← Choose Java SE 21 Archive

# Installing Java

- Choose installer compatible with your device:

macOS x64 DMG Installer	183.50 MB	<a href="https://download.oracle.com/java/21/archive/jdk-21.0.1_macos-x64_bin.dmg">https://download.oracle.com/java/21/archive/jdk-21.0.1_macos-x64_bin.dmg</a> (sha256)
Windows x64 Compressed Archive	185.39 MB	<a href="https://download.oracle.com/java/21/archive/jdk-21.0.1_windows-x64_bin.zip">https://download.oracle.com/java/21/archive/jdk-21.0.1_windows-x64_bin.zip</a> (sha256)
Windows x64 Installer	163.82 MB	<a href="https://download.oracle.com/java/21/archive/jdk-21.0.1_windows-x64_bin.exe">https://download.oracle.com/java/21/archive/jdk-21.0.1_windows-x64_bin.exe</a> (sha256)
Windows x64 msi Installer	162.60 MB	<a href="https://download.oracle.com/java/21/archive/jdk-21.0.1_windows-x64_bin.msi">https://download.oracle.com/java/21/archive/jdk-21.0.1_windows-x64_bin.msi</a> (sha256)

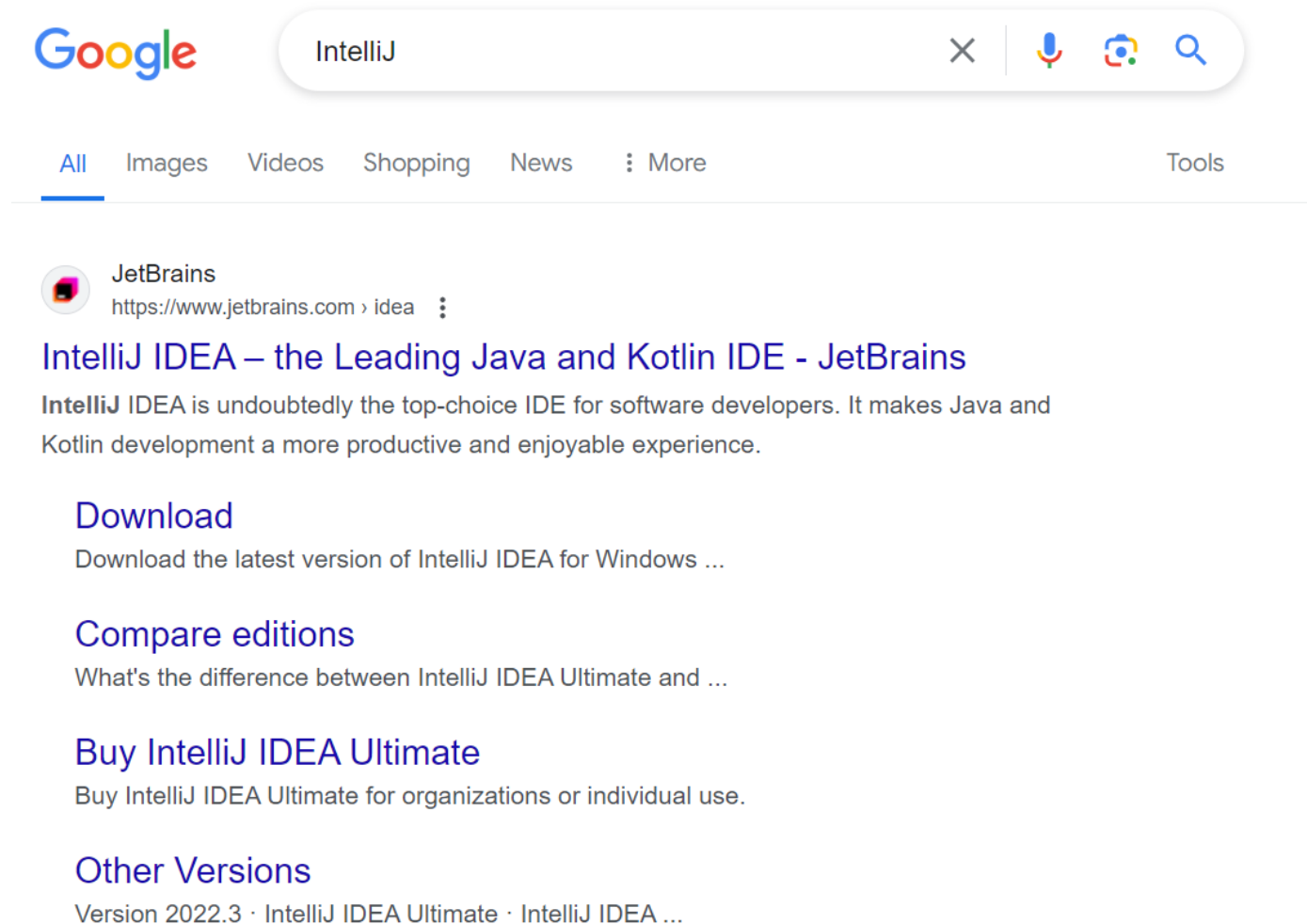
- Install:





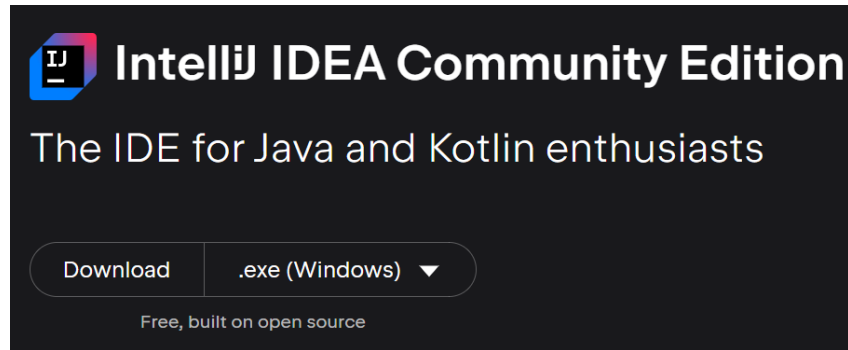
# Installing Java IDE

- Search for IntelliJ IDEA:

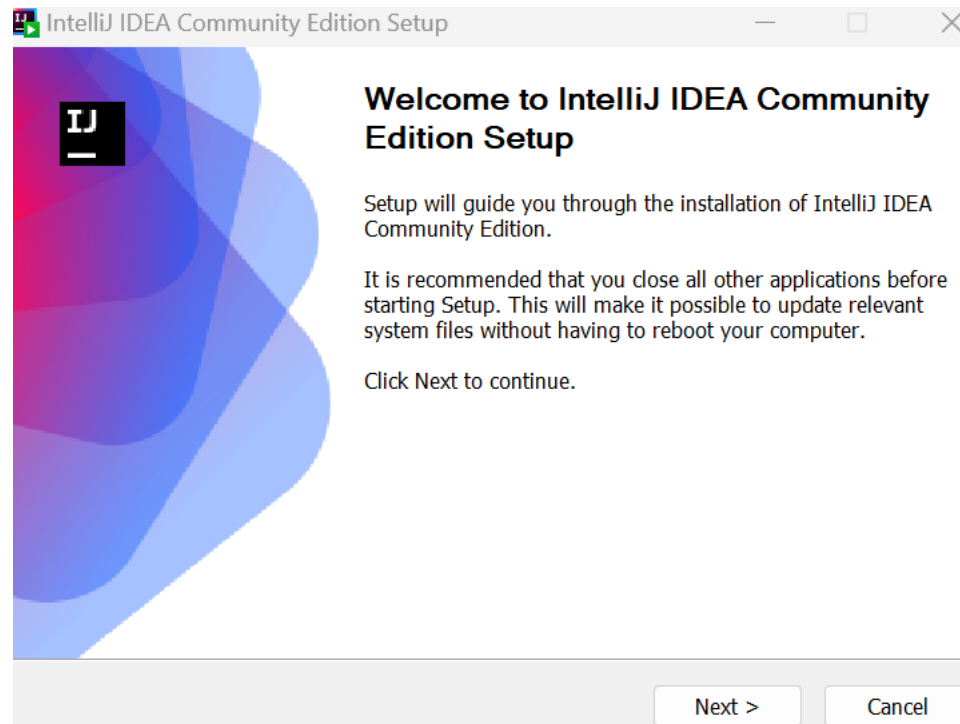


# Installing Java IDE

- Download Community Edition:




- Install:



# Installing Java IDE

- Setup of Environment Variables:



## Installation Options

Configure your IntelliJ IDEA Community Edition installation

Create Desktop Shortcut

☒ IntelliJ IDEA Community Edition

Update PATH Variable (restart needed)

☒ Add "bin" folder to the PATH

Update Context Menu

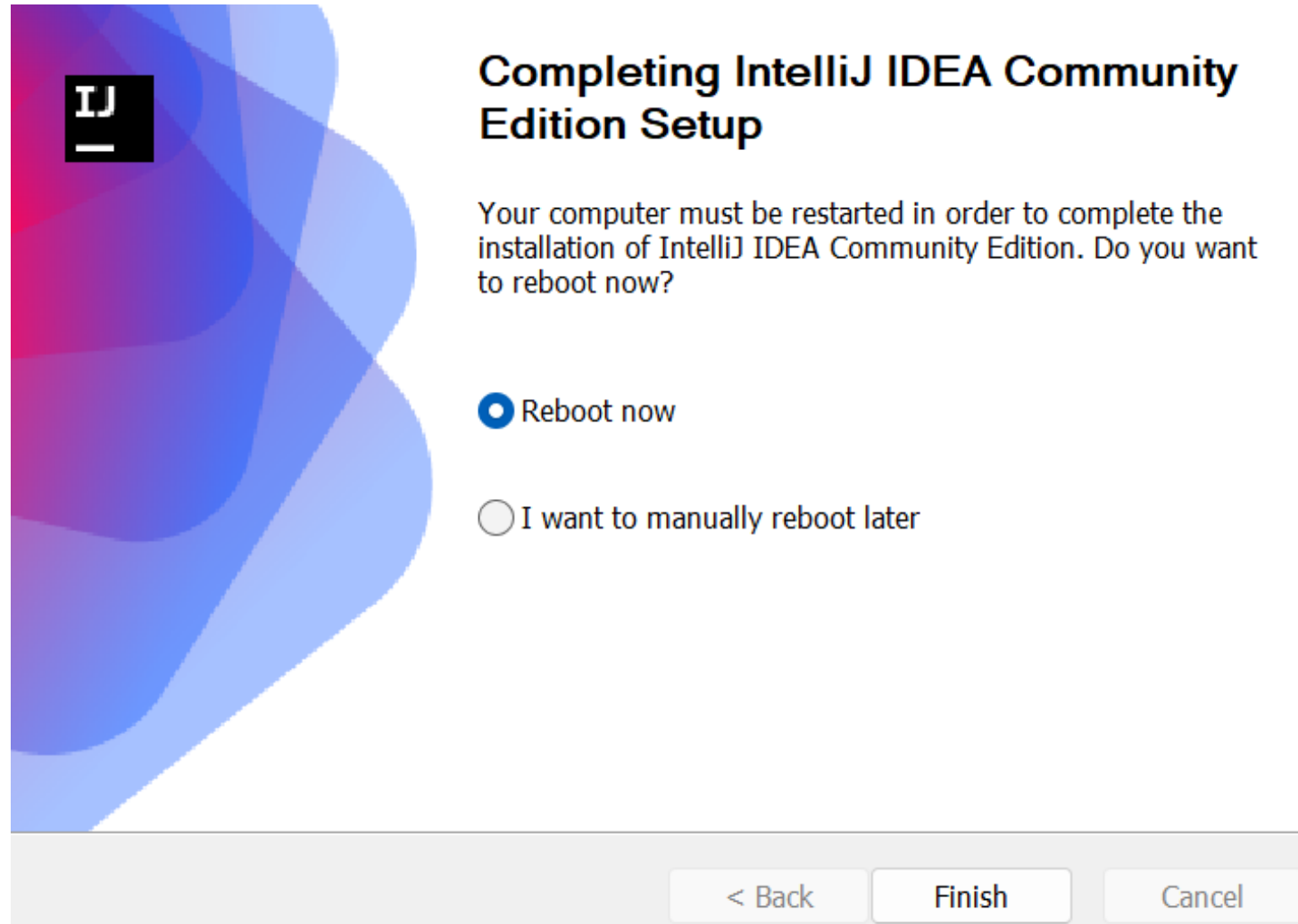
☒ Add "Open Folder as Project"

Create Associations

☒ .java   ☐ .gradle   ☐ .groovy   ☒ .kt   ☒ .kts   ☐ .pom

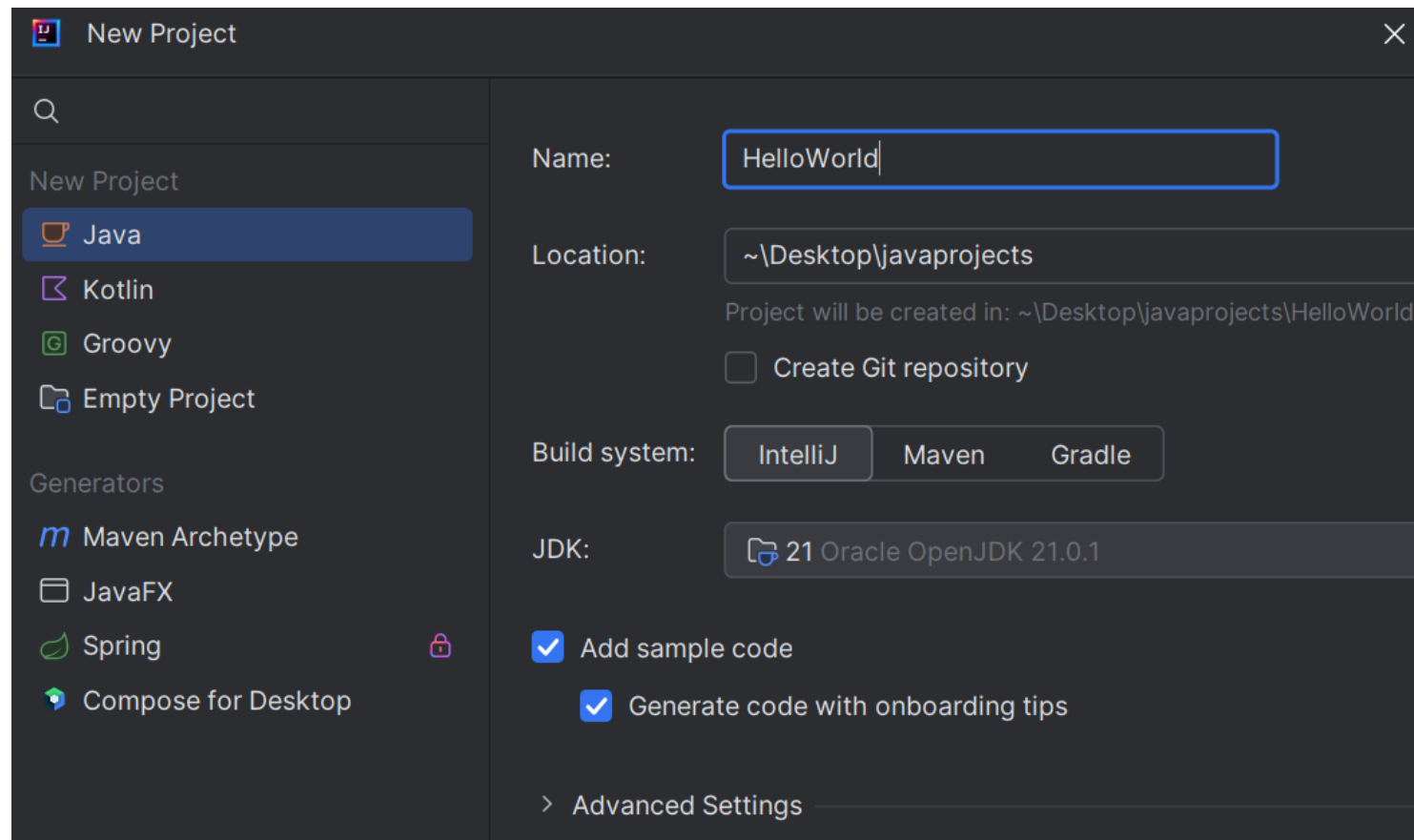
# Installing Java IDE

- Reboot your PC:



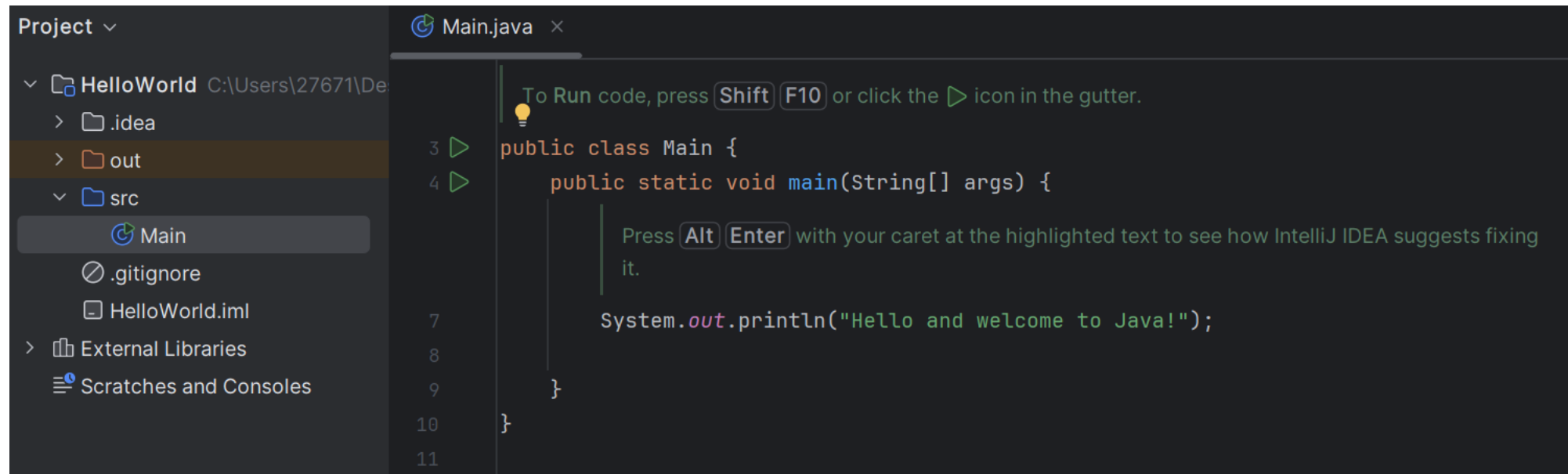
# Installing Java IDE

- Name your project > choose Location folder:

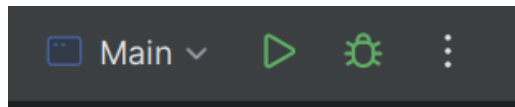


# Installing Java IDE

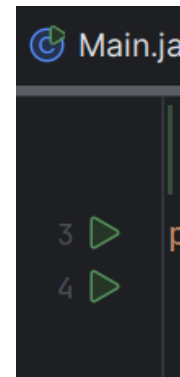
- First HelloWorld project:



- To Run, click:



Or:



Output:

```
"C:\Program Files\Java\jdk-21\bin\java.exe"  
Hello and welcome to Java!  
  
Process finished with exit code 0
```

# Java Cheat Sheet

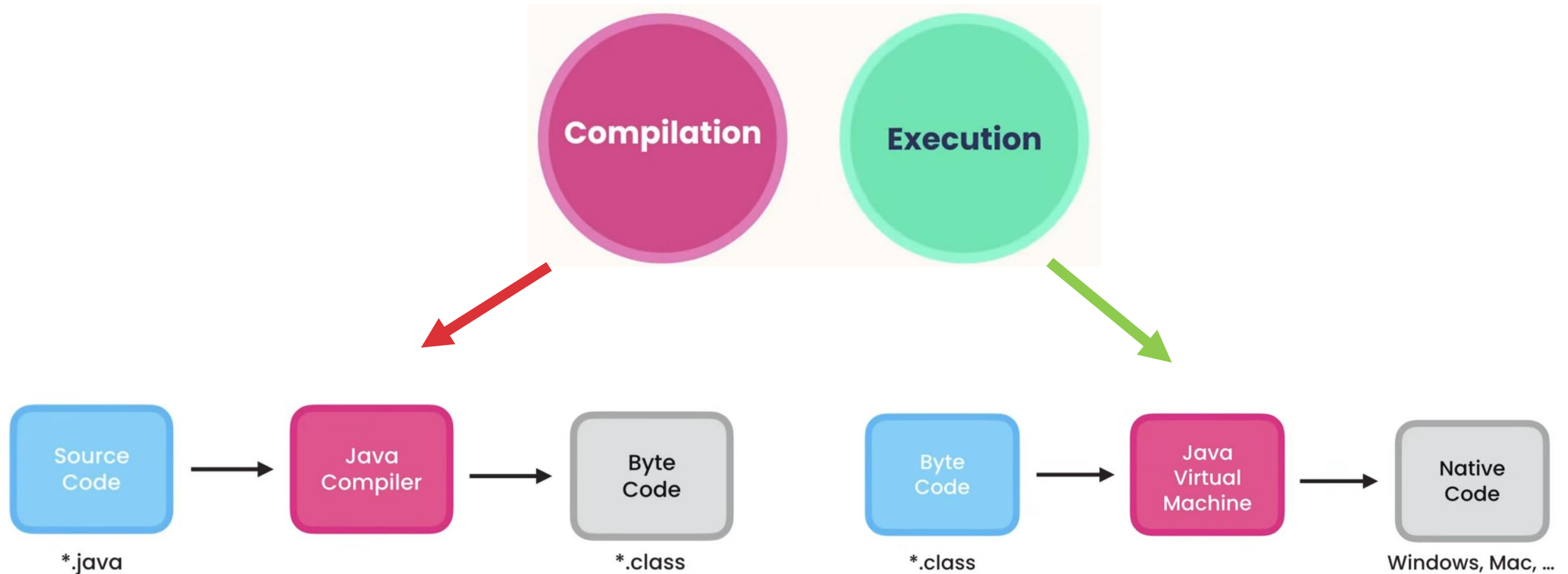
---

- Download the Java Cheat Sheet:

<https://codewithmosh.lpages.co/java-cheat-sheet/>

# Executing Java

- Running a Java application involves two main stages: compilation and execution.





# Executing Java

## Compilation

### 1) Source Code (.java):

- The Java application starts as human-readable source code files with a ".java" extension.
- This source code contains the instructions and logic written by the developer.

### 2) Compiler (javac):

- The source code is then compiled using the Java compiler (javac).
- The compiler translates the Java source code into bytecode.
- Bytecode is a set of instructions understood by the Java Virtual Machine (JVM).

### 3) Bytecode (.class):

- The output of the compilation process is one or more bytecode files with a ".class" extension.
- These bytecode files contain instructions in a platform-independent format.

# Executing Java

---

## Execution:

### 1) Java Virtual Machine (JVM):

- The JVM executes Java bytecode.
- The JVM is platform-dependent, meaning there are different implementations for Windows, Mac, Linux, etc. but they all adhere to the Java Virtual Machine Specification.

### 2) Interpreter and Just-In-Time (JIT) Compiler:

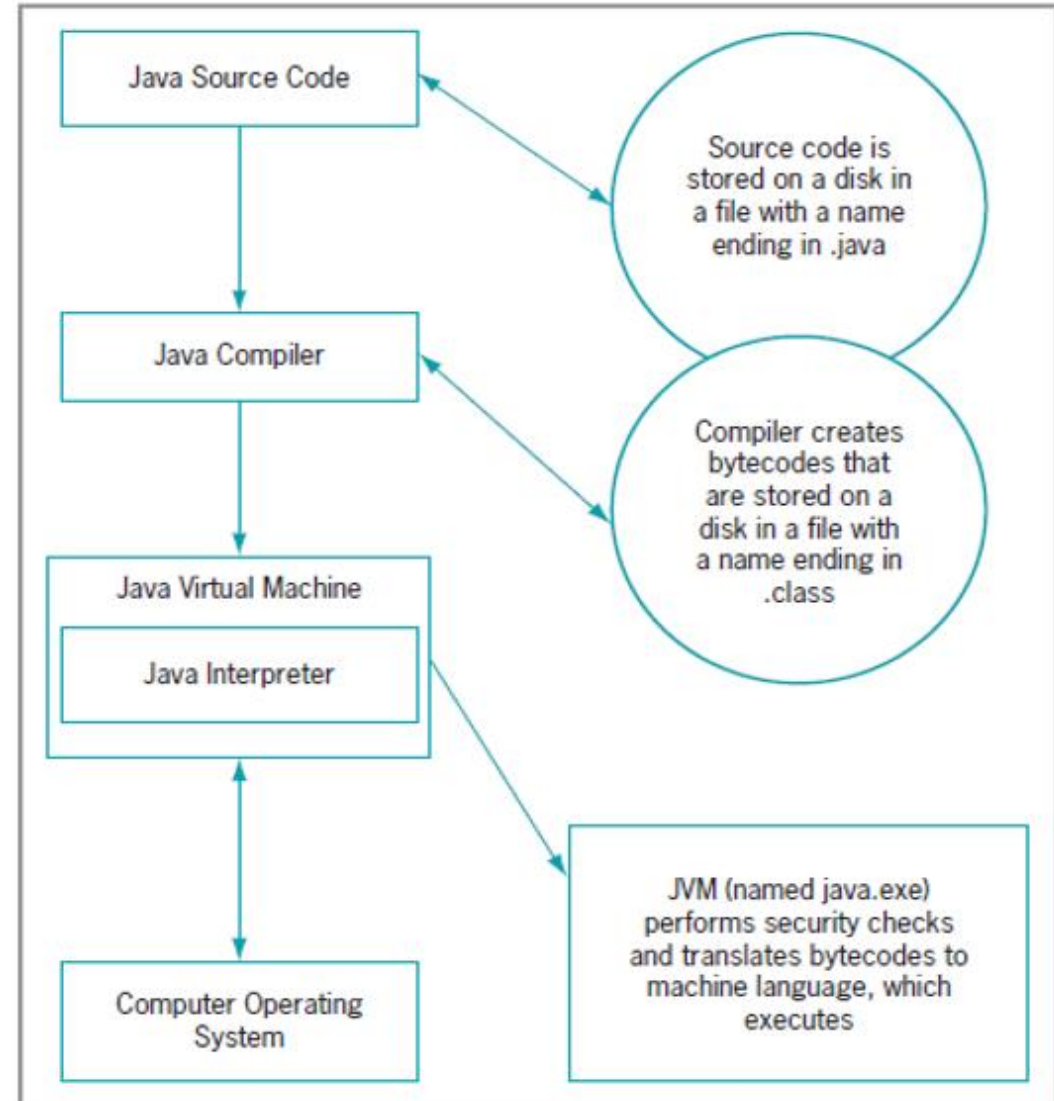
- JVM loads the bytecode and the JVM's interpreter reads and executes the bytecode.
- Uses Just-In-Time (JIT) compile which dynamically compiles bytecode into native machine code for the specific hardware it's running on, which is much faster than interpreting bytecode directly.

### 3) Native Code Execution:

- The JIT compiler generates native machine code specific to the hardware and operating system.
- This native code is executed directly by the processor, whether it's on a Windows, Mac, Linux, or any other supported environment.

# The Java Environment

- Java runs on a hypothetical computer known as the Java Virtual Machine (JVM).



# Java Program Types

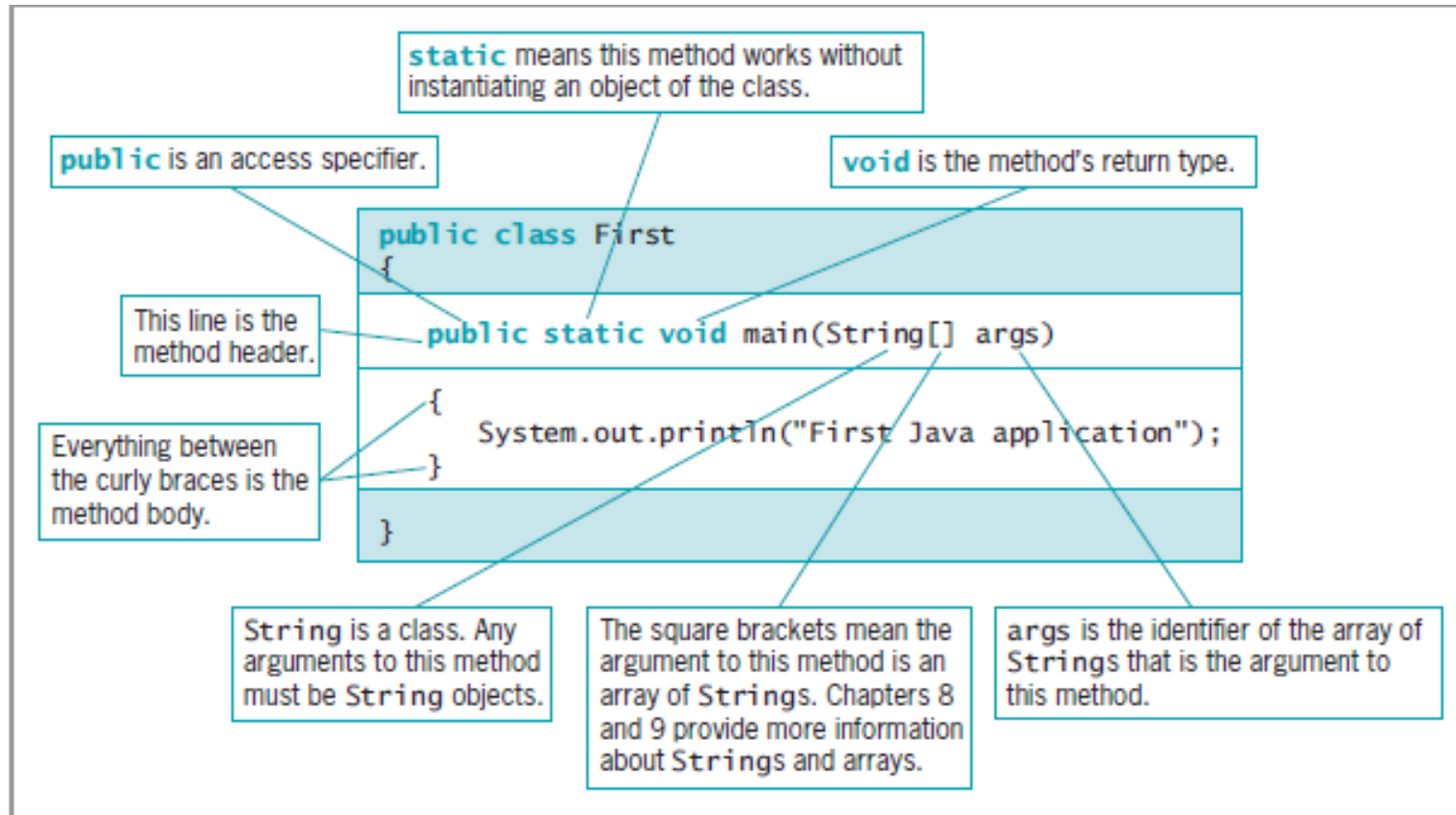
---

You can write different kinds of programs using Java:

- 1) **Desktop Applications** - Graphical User Interface (GUI) Applications using libraries like Swing and JavaFX.
- 2) **Web Applications** - Dynamic Web Applications like Servlets, JSP, and frameworks like Spring and JSF.
- 3) **Mobile Applications** - Apps for Android devices, developed using the Android SDK.
- 4) **Enterprise Applications** - Large-scale, complex systems for businesses, including CRM systems, ERP systems, and inventory management, using Java EE (Jakarta EE) technologies.
- 5) **Server-Side Applications** - Backend applications running on servers, including RESTful APIs, microservices, and high-performance server-side applications.

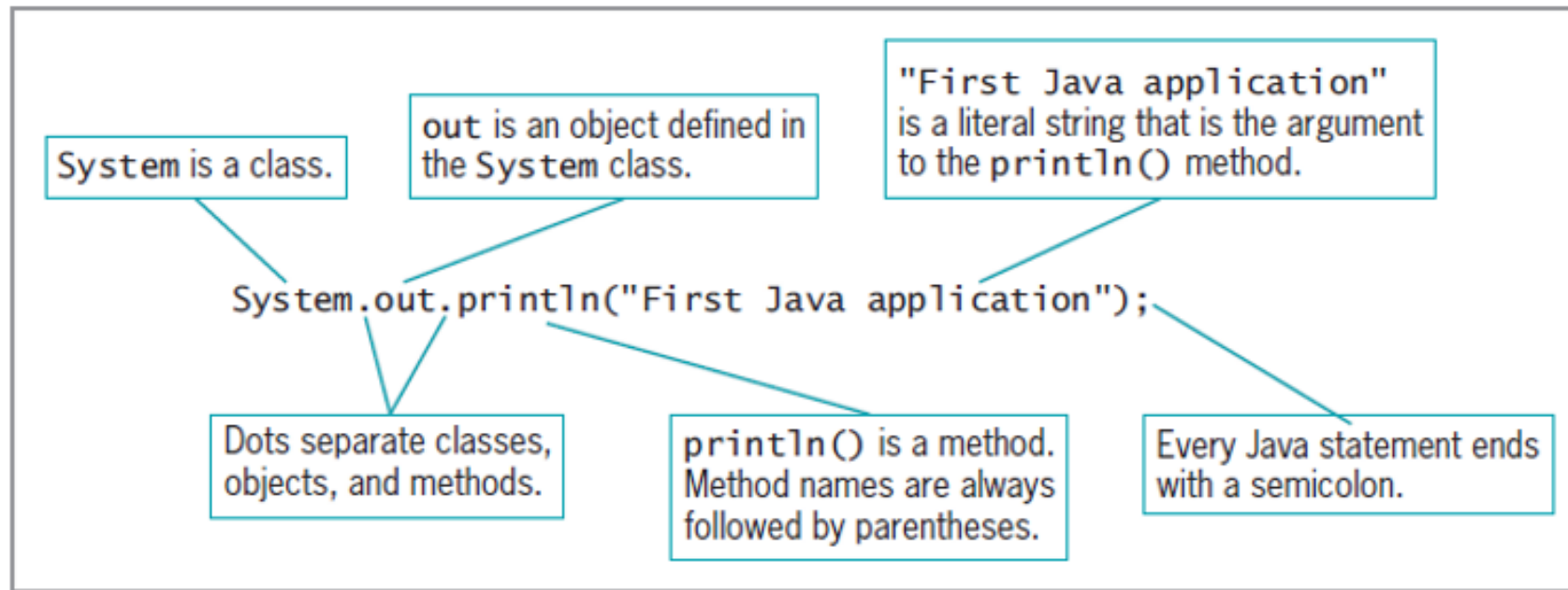
# Java Console Applications

- Anatomy of Java program:



# Java Console Applications

- Anatomy of Java statement:



# Java Variables

- Temporary storage space for a value (number, word etc.)

- Variable Declaration:

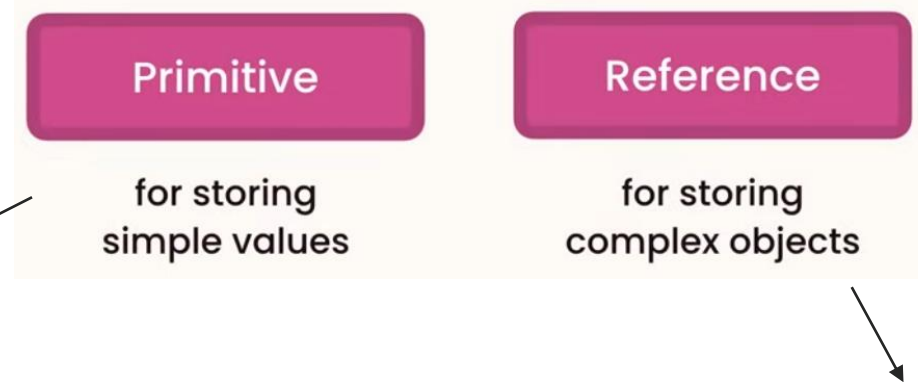
```
//Variable declaration  
int MyAge;  
myAge = 30;  
  
//Variable initialisation  
string surname = "Zuma";
```

- Variable Initialization:

```
//Variable initialisation  
string surname = "Zuma";
```

# Java Types

- Two examples exists:



Type	Bytes	Range
byte	1	[-128, 127]
short	2	[-32K, 32K]
int	4	[-2B, 2B]
long	8	
float	4	
double	8	
char	2	A, B, C, ...
boolean	1	true / false

Type	Explanation
Classes	Classes are reference types because when you create an object using a class, you're creating a reference to that object, not the object itself.
Interfaces	Interfaces are reference types because they define a contract or a type that classes can implement, creating a reference of that interface type.
Arrays	Arrays are reference types because they hold references to objects, allowing dynamic memory allocation and manipulation of complex data structures.
Enums	Enums are reference types because they represent a fixed set of named constants, and when you use an enum type, you're using a reference to one of its values.
Date	The <code>Date</code> class is a reference type because when you create a <code>Date</code> object, you're creating a reference to a specific instant in time, not the actual time itself.
String	The <code>String</code> class is a reference type because it represents a sequence of characters stored in memory, and when you create a <code>String</code> object, you're creating a reference to it.



# Java Primitive Types

- Declaring different types:

```
//Java Types

int age = 30;    //Integers
//Use Underscore to separate every 3 digits for readability
int population1 = 2_134_567_867; //maximum value: 2,147,483,647
long population2 = 2_456_890_234L; //Any value above is long
String fName = "Simba";
boolean trueOrFalse = true;

double price = 2500.99; //All decimals are double by default
float temp = 30.50F;    //Use suffix to represent the decimal
char letter = 'A';

//Output
System.out.println("Price is: " + price + " - Age is: " + age);
String message = String.format("Population is: %d", population1);
System.out.println(message);
```

# Formatting Strings

- The *String.format()* method in Java allows you to create formatted strings by replacing placeholders with values. It provides a flexible way to control the appearance of the output.

```
String formattedString = String.format(format, args);
```

- *format*: A format string containing placeholders for values. Placeholders are marked by % followed by a format specifier (e.g., %s, %d, %f).
- *args*: Optional arguments to replace the placeholders in the format string.

## Format Specifiers:

- `%s`: String
- `%d`: Integer
- `%f`: Floating-point number
- `%b`: Boolean
- `%c`: Character
- `%n`: Newline
- `%t`: Date/time (various sub-specifiers)

# Java Reference Types

- Reference types are data types that store references (memory addresses) to objects rather than the actual object data itself.

```
//Java Reference Types
//Date
Date now = new Date();
System.out.println(now);    //Shortcut sout + tab

//String
String message = new String(original: "Belgium Campus");
System.out.println(message.toUpperCase());
```

‘new String()’ is redundant

# Java Reference Types

- *'new String()' is redundant* - using new String() to create a new String object is considered redundant in most cases.
- This is because Java provides a convenient string literal syntax, where you can directly assign a string value enclosed in double quotes to a String variable without using the new keyword:

```
String str1 = "Hello"; // Using string literal syntax  
String str2 = new String("Hello"); // Using new String() - redundant
```

- Both *str1* and *str2* will contain the string "Hello", but the first approach is simpler and more efficient.
- It is recommended to use string literals instead of new String() to create String objects in Java.
- Since strings are object reference type in Java, they come with a variety of methods for manipulating and working with strings.

# String Methods in Java

Method	Description
<code>`charAt(int index)`</code>	Returns the character at the specified index within the string.
<code>`length()`</code>	Returns the length of the string.
<code>`substring(int beginIndex)`</code>	Returns a substring of the string starting from the specified index.
<code>`substring(int beginIndex, int endIndex)`</code>	Returns a substring of the string starting from the <code>`beginIndex`</code> and ending before the <code>`endIndex`</code> .
<code>`indexOf(String str)`</code>	Returns the index of the first occurrence of the specified substring within the string, or -1 if not found.
<code>`lastIndexOf(String str)`</code>	Returns the index of the last occurrence of the specified substring within the string, or -1 if not found.
<code>`startsWith(String prefix)`</code>	Returns <code>`true`</code> if the string starts with the specified prefix, otherwise returns <code>`false`</code> .

# String Methods in Java

Method	Description
<code>`endsWith(String suffix)`</code>	Returns <code>`true`</code> if the string ends with the specified suffix, otherwise returns <code>`false`</code> .
<code>`toLowerCase()`</code>	Converts all characters in the string to lowercase.
<code>`toUpperCase()`</code>	Converts all characters in the string to uppercase.
<code>`trim()`</code>	Removes leading and trailing whitespace from the string.
<code>`replace(char oldChar, char newChar)`</code>	Replaces all occurrences of <code>`oldChar`</code> in the string with <code>`newChar`</code> .
<code>`replaceAll(String regex, String replacement)`</code>	Replaces all substrings that match the given regular expression <code>`regex`</code> with the specified replacement string.
<code>`split(String regex)`</code>	Splits the string into an array of substrings based on the given regular expression <code>`regex`</code> .

# Input and Output in Java

- In Java, you can perform input and output operations using the console using:
- the *System.in* stream
- the *System.out* stream

```
Main.java x
1
2 import java.util.Scanner; //Imports Scanner class
3
4 public class Main {
5     public static void main(String[] args) {
6
7         Scanner scanner = new Scanner(System.in);
8
9         System.out.print("Enter your name: ");
10        String name = scanner.nextLine();
11
12        System.out.print("Enter your age: ");
13        int age = scanner.nextInt();
14
15        System.out.println("Hello, " + name + "! You are " + age + " years old.");
16
17        scanner.close(); // Remember to close the scanner when done
18
19    }
20 }
```

# Console Output Formatting:

- You can use formatting options with *System.out.printf()* for more control over output formatting.

```
public class Main {  
    public static void main(String[] args) {  
  
        String name = "John";  
        int age = 30;  
        System.out.printf("Hello, %s! You are %d years old.\n", name, age);  
  
    }  
}
```



# Exercises

---

1. What is the character at index 3 in the string : *String str = "Hello World";*
2. What is the length of the string: *String str = "Java Programming";*
3. What is the substring of "Hello World" starting from index 6?
4. What is the substring of "Hello World" from index 3 to index 7?
5. What is the index of the first occurrence of "world" in the string:  
*String str = "Hello world, welcome to the world";*

# Exercises

---

6. What is the index of the last occurrence of "the" in the string:

```
String str = "The quick brown fox jumps over the lazy dog";
```

7. Does the string "Java Programming" start with "Java"?

8. Remove leading and trailing whitespace from the string " Java Programming "

9. Replace all occurrences of 'o' with 'x' in the string "Hello World".

10. Split the string "Java Programming is fun" into an array of words.

# Exercises

11. What are Features of Java Programming?
12. Differentiate between Java Applets and Applications?
13. Explain the Importance of JVM.
14. What is the role of Java in Internet?
15. Create a console Java App that outputs the following:

```
Who let the dogs out...???  
*****  
*****  
Hoo, Hoo, Hoo hoo hoo!!!  
*****
```

Thank You!

*The End*

info@belgiumcampus.ac.za

+27 10 593 53 68



/belgiumcampusSA



#Belgium Campus



/belgiumcampus