Vietnam National University of HCMC
International University
School of Computer Science and Engineering

Introduction to Java (IT069IU)

Nguyen Trung Ky, PhD

ntky@hcmiu.edu.vn

it.hcmiu.edu.vn/user/ntky

Previously,

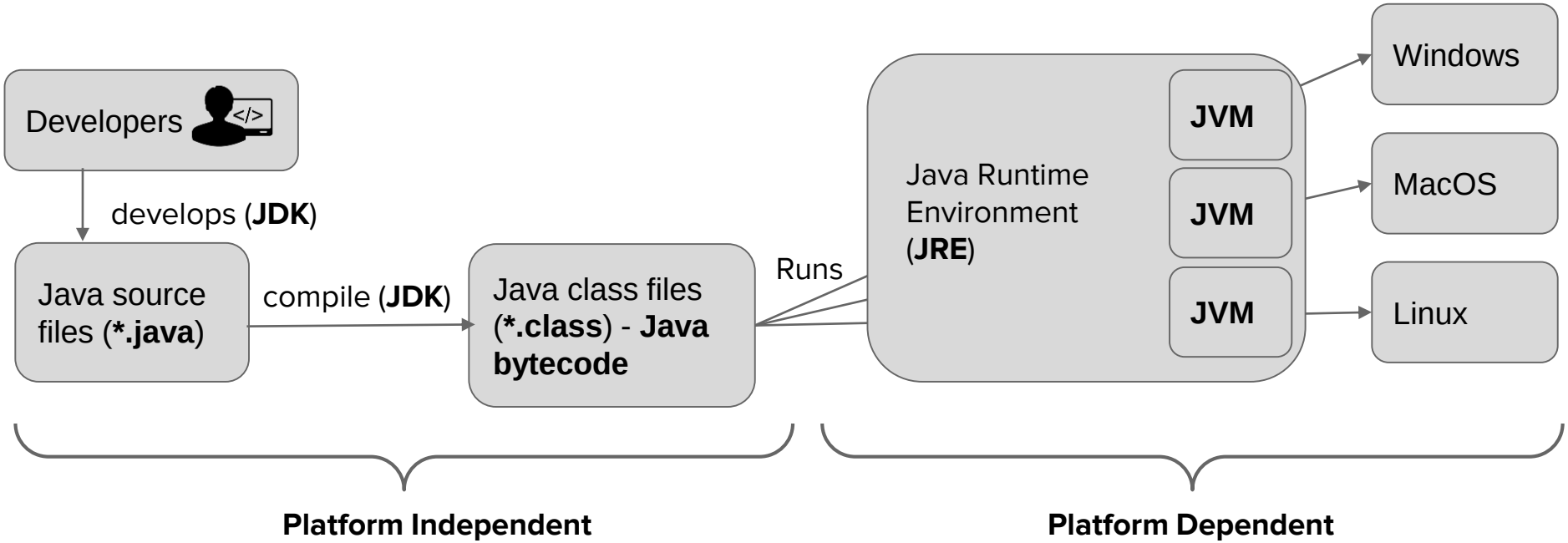
We talked about:

- Programming Paradigms.
- Basic idea of Object Oriented Programming (OOP).
- The importance of Java.
- The differences between JDK, JRE, JVM and how these works together.

Agenda

- Different Java Platforms
- Our choice of JDK
- The best IDE selections
- Create the first Java programs
- Compile and Run with commands or on Eclipse
- Java data types:
  - Primitive
  - Non-primitive (reference types)
- Variable
- Operators

Revisited: How Java works (JDK, JRE, JVM)



=> Java is **cross platform** - “Write once, run everywhere!”

Cross-platform = "Write Once, Run Anywhere"

Devices		
Airplane systems	ATMs	Automobile infotainment systems
Blu-ray Disc™ players	Cable boxes	Copiers
Credit cards	CT scanners	Desktop computers
e-Readers	Game consoles	GPS navigation systems
Home appliances	Home security systems	Light switches
Lottery terminals	Medical devices	Mobile phones
MRIs	Parking payment stations	Printers
Transportation passes	Robots	Routers
Smart cards	Smart meters	Smartpens
Smartphones	Tablets	Televisions
TV set-top boxes	Thermostats	Vehicle diagnostic systems

Fig. 1.1 | Some devices that use Java.

The Java Platforms

- **Java Standard Edition (Java SE)**
  - Developing desktop and server applications.
  - This course will mainly focus on this edition.

- **Java Enterprise Edition (Java EE)**
  - Developing large-scale, distributed networking applications and web-based applications.
  - You will learn in Web Application Development course (IT093IU).

- **Java Micro Edition (Java ME)**
  - A subset of Java SE
  - Developing applications for resource-constrained embedded devices, such as smartwatches, MP3 players, television set-top boxes, smart meters (for monitoring electric energy usage).

Our choice for Java SDK

- Oracle JDK 17
  - LTS version (Long-Term Support)
  - Released in 09/2021 and got supported till 09/2029
  - The implementation of Java Standard Edition (Java SE)

Java SE Development Kit 17.0.2 downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

Linux **macOS** **Windows**

Product/file description	File size	Download
x64 Compressed Archive	171.34 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256)
x64 Installer	152.43 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256)
x64 MSI Installer	151.32 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256)

https://www.oracle.com/java/technologies/downloads/#jdk17-windows

The best Java IDEs

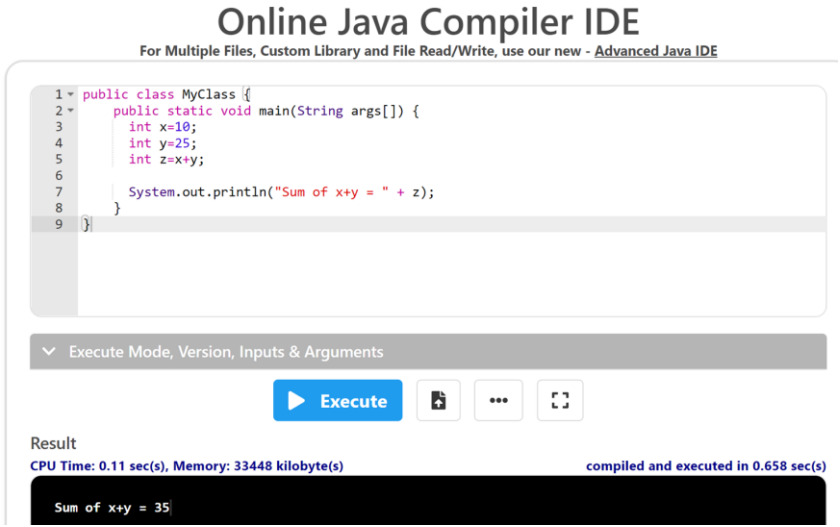
- **IDE is** Integrated Development Environment (editor + compiler + debugger + convenient tools + GUI)
- **IntelliJ IDEA** (Recommended)
  - A commercial IDEs developed by JetBrains.
  - Students can get Ultimate version: https://www.jetbrains.com/community/education/#students
  - Link to download: https://www.jetbrains.com/idea/
- **Netbeans**
  - Open-source and free.
  - Link to download: http://www.netbeans.org/index.html.
- **Eclipse**
  - Developed by the eclipse open-source community.
  - Link to download: http://www.eclipse.org.

IntelliJ IDEA

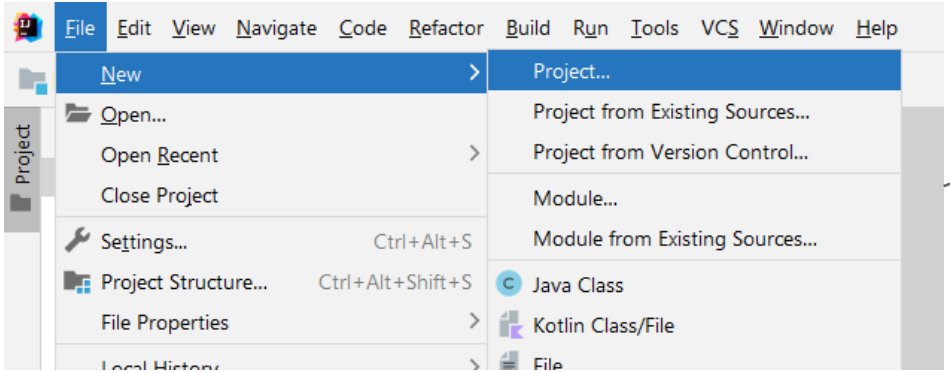
Apache NetBeans

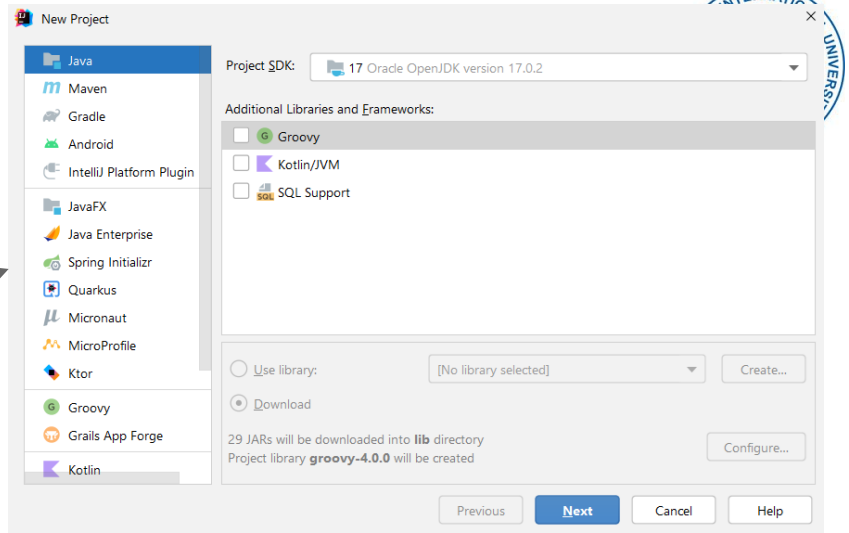
An instant-noodle web IDEs

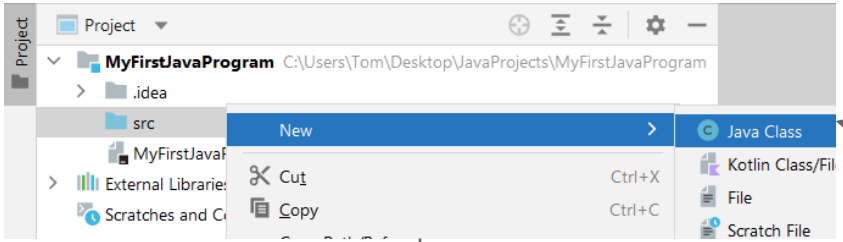
- For people who haven't installed SDK or IDEs on your laptop and still want to follow the lecture and practice Java at home.
- You can try the online Java compiler to run Java right on your browser: https://www.jdoodle.com/online-java-compiler/
- This only support only one main class which is extremely limited so you still need a full IDEs on your laptop for your project and labs.

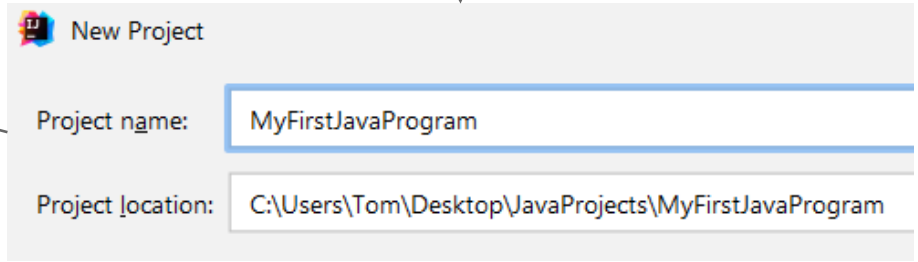


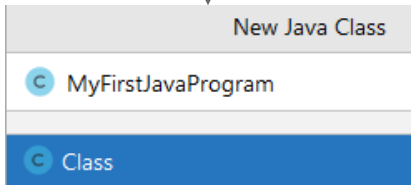
First Java Project on IntelliJ



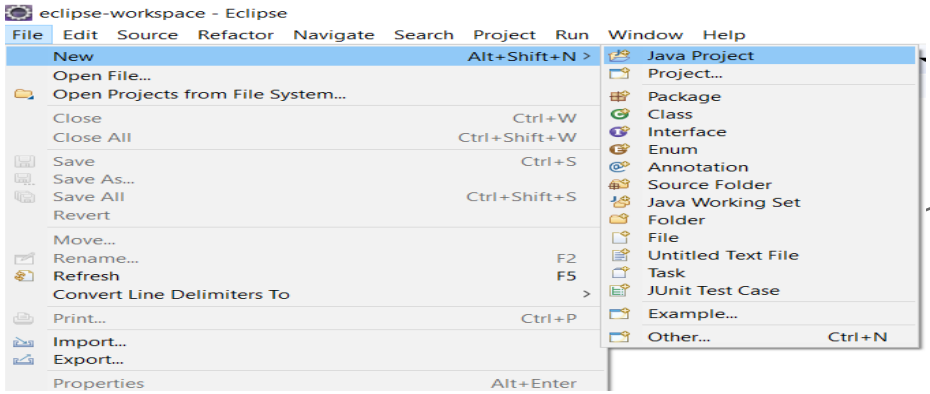


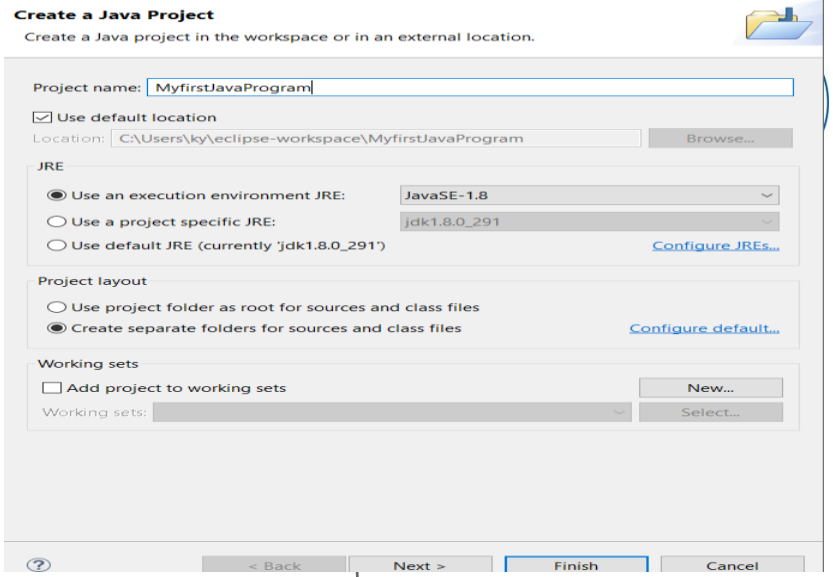


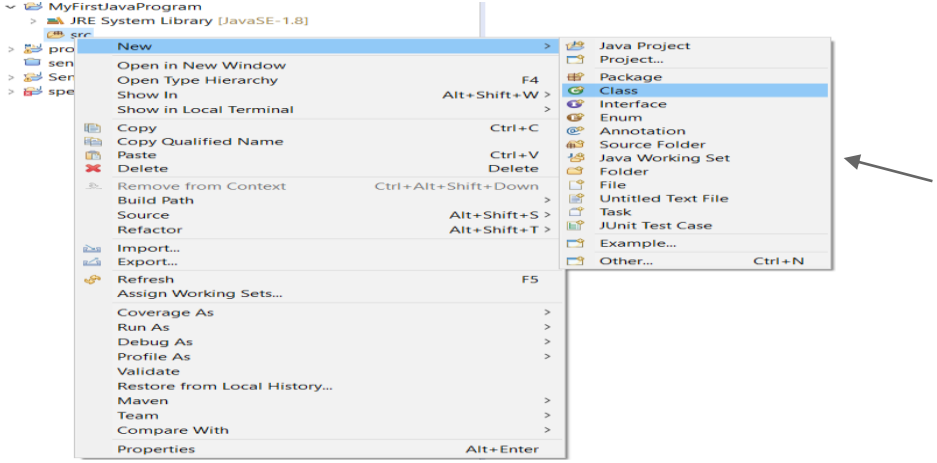


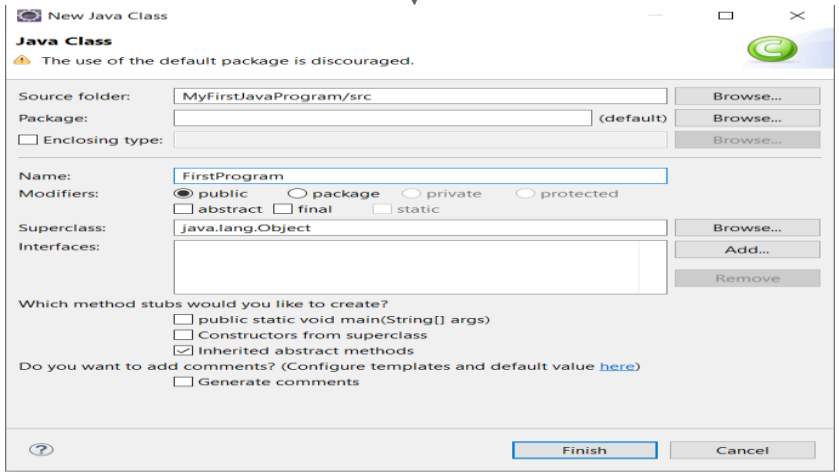


First Java Project on Eclipse









Our first Java Program

MyFirstJavaProgram.java

```java
public class MyFirstJavaProgram {
    public static void main(String[] args) {

        // This will print out the string "Hello World"
        System.out.println("Hello World");

    }
}
```

Let's compile

- First, we check if our JDK is installed correctly

```
C:\Users\ky\eclipse-workspace>cd MyFirstJavaProgram

C:\Users\ky\eclipse-workspace\MyFirstJavaProgram>java --version
java 10.0.1 2018-04-17
Java(TM) SE Runtime Environment 18.3 (build 10.0.1+10)
Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10.0.1+10, mixed mode)
```

- We can compile the java file by the command "javac"

```
C:\Users\ky\eclipse-workspace\MyFirstJavaProgram\src>javac FirstProgram.java
```

- The compiler creates a new compiled file with the extension .class, containing java bytecode

```
09/06/2022  06:22 PM               430 FirstProgram.class
09/06/2022  03:26 PM               169 FirstProgram.java
               2 File(s)            599 bytes
               2 Dir(s)  34,720,440,320 bytes free
```

Let's run

- To run the compiled .class file, we need to use the command "java"

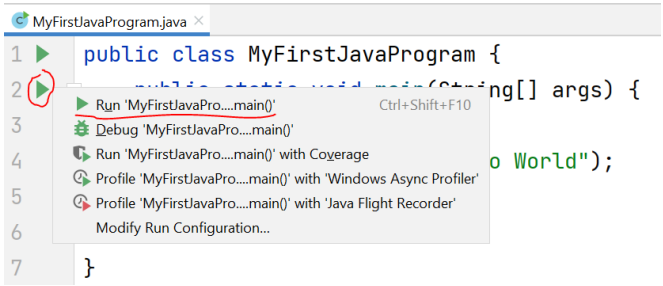
```
C:\Users\ky\eclipse-workspace\MyFirstJavaProgram\src>java FirstProgram
Hello World!
```

Awesome! Now we can see the output string "Hello World"

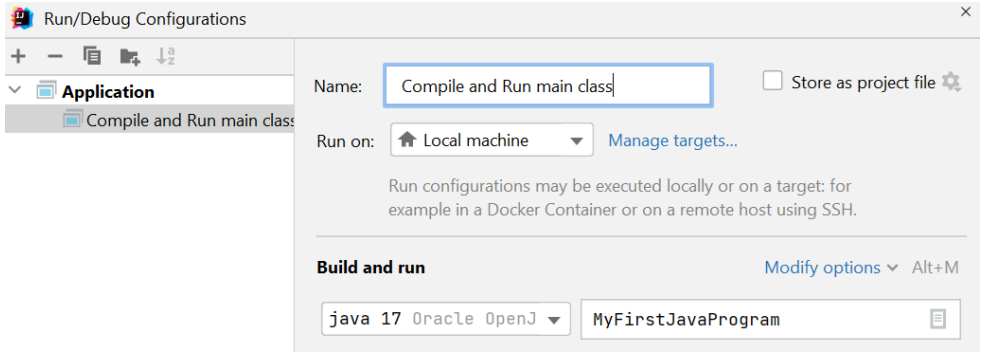
When your code outputs "Hello World!"

Run & Compile Automatically on IntelliJ

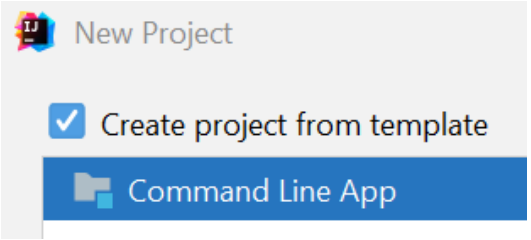
Method 1: Click on the main method

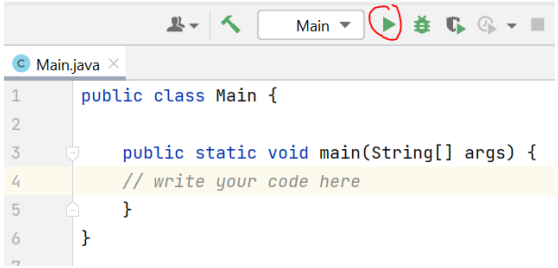


Method 2: Setup a build configuration for the project

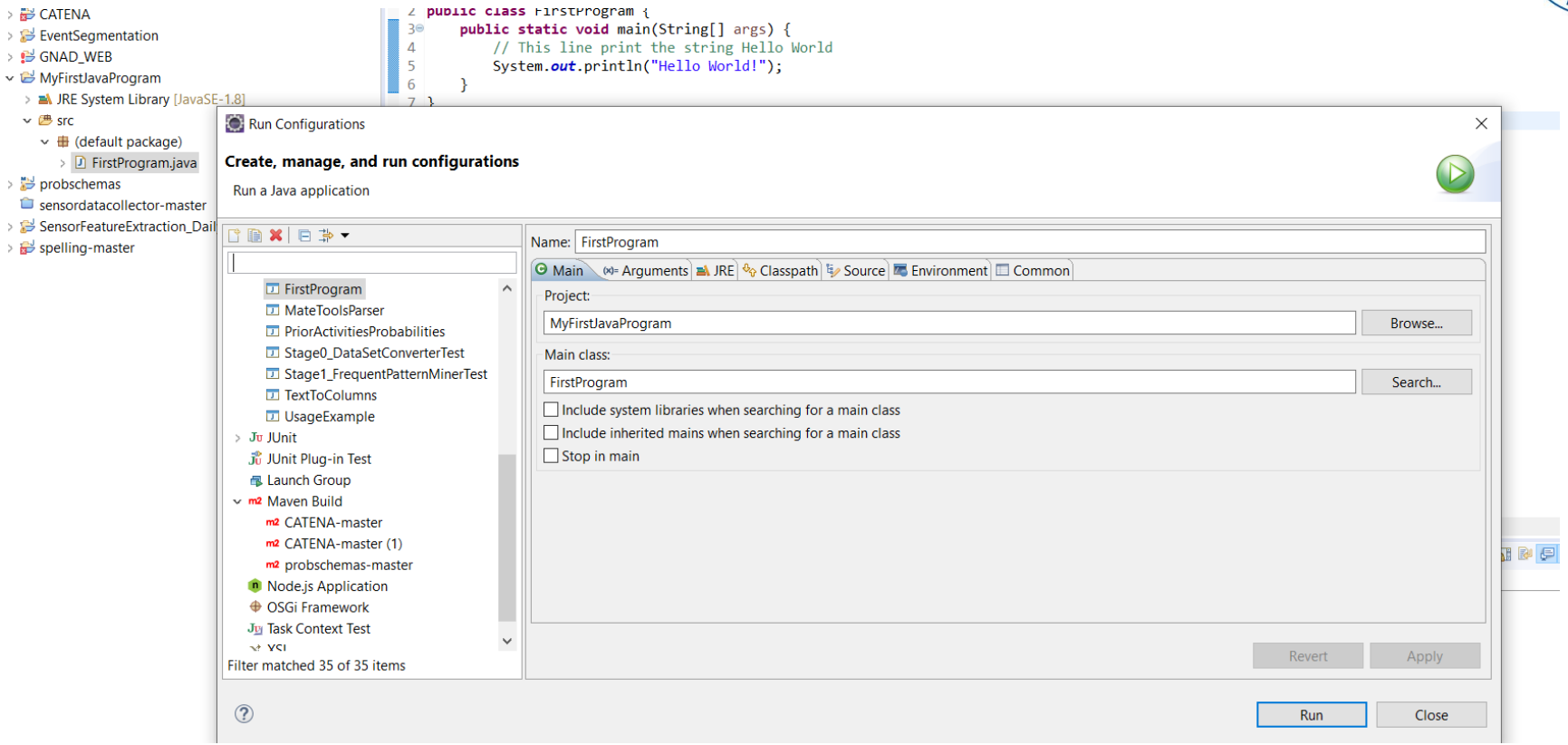


Method 3: Setup the new project from the template





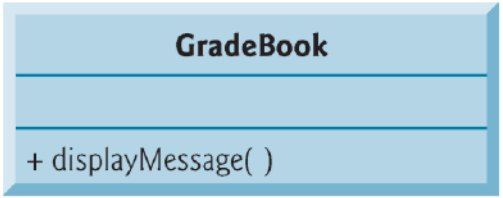
Run & Compile Automatically on Eclipse



Let's try it with Multiple Class Files

GradeBook.java

```
1  // Fig. 3.1: GradeBook.java
2  // Class declaration with one method.
3
4  public class GradeBook
5  {
6     // display a welcome message to the GradeBook user
7     public void displayMessage()
8     {
9        System.out.println( "Welcome to the Grade Book!" );
10    } // end method displayMessage
11 } // end class GradeBook
```



UML class diagram

Performs the task of displaying a message on the screen; method `displayMessage` must be called to perform this task

Fig. 3.1 | Class declaration with one method.

Let's try it with Multiple Class Files

GradeBookTest.java

```
1  // Fig. 3.2: GradeBookTest.java
2  // Creating a GradeBook object and calling its displayMessage method.
3
4  public class GradeBookTest
5  {
6     // main method begins program execution
7     public static void main( String[] args )
8     {
9        // create a GradeBook object and assign it to myGradeBook
10       GradeBook myGradeBook = new GradeBook();
11
12       // call myGradeBook's displayMessage method
13       myGradeBook.displayMessage();
14    } // end main
15 } // end class GradeBookTest
```

Creates a `GradeBook` object and assigns it to variable `myGradeBook`

Invokes method `displayMessage` on the GradeBook object that was assigned to variable `myGradeBook`

```
Welcome to the Grade Book!
```

Fig. 3.2 | Creating a `GradeBook` object and calling its `displayMessage` method.

Compiling & Run with Multiple Class Files

- To compile multiple files, just list all the names of java files:

  javac GradeBook.java GradeBookTest.java

- Or to compile all java files in the directory:

  javac *.java

- After the compilation, two new compiled file with the extension class are created:

  Gradebook.class, GradeBookTest.class

- To run a particular compiled file containing the main method, in our case, GradeBookTest.class:

  java GradeBookTest

- You should see the output message:

```
Welcome to the Grade Book!
```

Types of Comment Lines

There are 3 types of comments:

- Single line comment //
- Multiple line comment /* */
- Javadoc comment /** */
  - prepare program documentation in HTML format.

```
// Text-printing program.
```

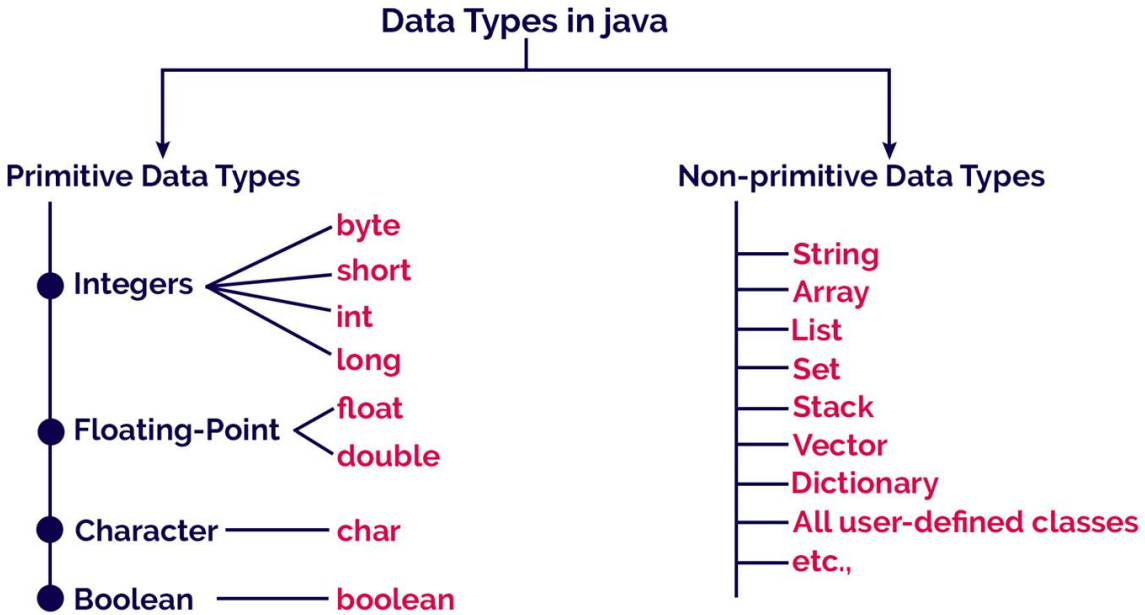
```
/* This is a traditional comment. It
   can be split over multiple lines */
```

```
1  public class CommentsInJava {
2      /** Javadoc comment...appropriate for documenting a class or method for javadoc utility.
3          Example: The main method is run automatically by the Java Virtual Machine */
4      public static void main (String[] args) {
5          int i=100;
6          // Single line comment...appropriate for documenting a statement.
7          // Example: The following statement prints a string to the console:
8          System.out.println("This is the first statement after the comment!");
9
10         /* Multi-line comment...appropriate for commenting code that is
11            not needed currently but may be
12            necessary in the future.
13         Example: System.out.println("This is the integer variable I created: " + i);
14         System.out.println("This is the last statement in the block!");
15         */
16     }
17 }
```

Data Types

Data Types

- Java is statically typed and also a strongly typed language.
- There are two categories of data types:
  - **Primitive** data types (8)
  - **Non-primitive** data types (**reference** type)



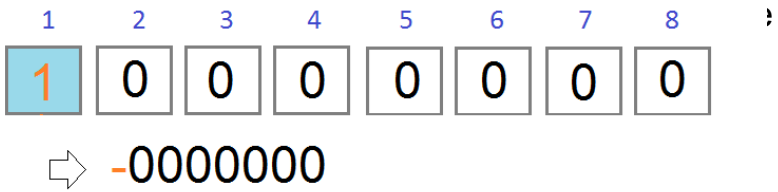
Byte

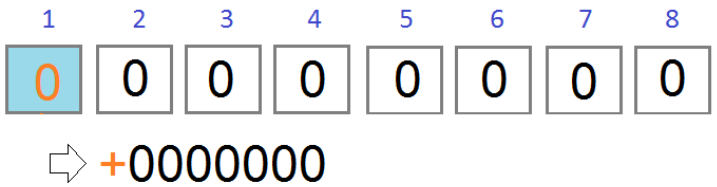
Byte data type is an **8-bit** signed two's complement integer.

- Minimum value is -128 (-2^7)
- Maximum value is 127 (2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- They are also useful when you're working with raw with Java's other built-in types
- Example:

byte a = 100;

byte b = -50;





Short

Short data type is a **16-bit** signed two's complement integer.

- Minimum value is –32,768 (-2^15)
- Maximum value is 32,767 (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int
- Default value is 0.
- Example:

```
short s= 10000;

short r = -20000;
```

Int

int data type is a **32-bit** signed two's complement integer.

- Minimum value is - 2,147,483,648 (-2^31)
- Maximum value is 2,147,483,647 (2^31 -1)
- Int is generally used as the **default data type for integral values** unless there is a concern about memory.
- The default value is 0.
- Example:

```
int a = 100000;

int b = -200000;
```

Long

Long data type is a **64-bit** signed two's complement integer.

- Minimum value is -9,223,372,036,854,775,808 (-2^63)
- Maximum value is 9,223,372,036,854,775,807 (2^63 -1)
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example:

```
long a = 100000L;

long b = -200000L;
```

Float

Float data type is a **single-precision 32-bit**

- Evaluating expressions that require fractional precision, such as square root, or transcendentals such as sine and cosine.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- It will become imprecise when the values are either very large or very small.
- Can be useful when representing dollars and cents.
- Example:

```
float houseTemperature = 23.5f;
```

Double

Double data type is a **double-precision 64-bit**

- This data type is generally used as **the default data type for decimal values**, generally the default choice.
- Double values should never be used for precise values such as currency. (use BigDecimal instead)
- The range is –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- Default value is 0.0d.
- Example:

```
double myWallet = 123.435;
```

Boolean

boolean data type represents **one bit** of information.

- There are only two possible values: **true** and **false**.
- This data type is used for checking true/false for if statement conditions.
- Default value is false.
- Example:

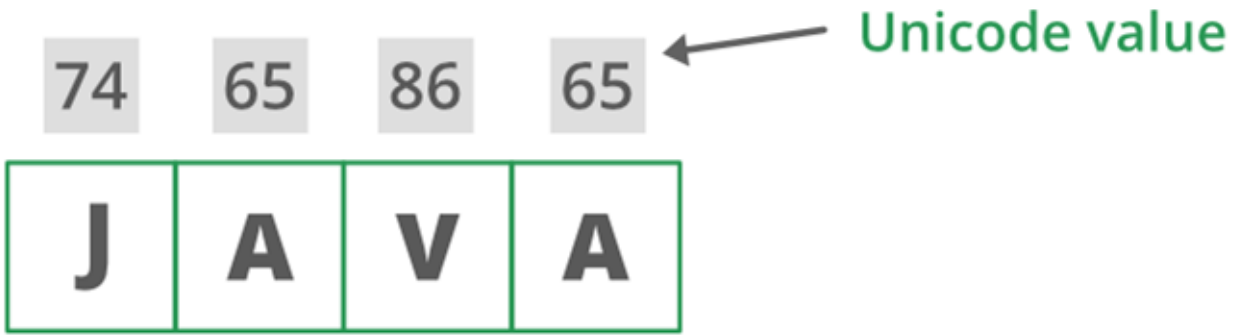
```
boolean one = true;

boolean larger = 8 > 4;
```

Char

char data type is a single **16-bit Unicode character**.

- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535).
- Char data type is used to store any character.
- Example:

  char letter = 'A';



https://en.wikipedia.org/wiki/UTF-16

Non-primitive data types (reference types)

- Non-primitive data types are called reference types because they refer to objects.
- The main difference between primitive and non-primitive data types are:
  - Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String)
  - Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot
  - A primitive type has always a value, while non-primitive types can be null.
  - A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
  - The size of a primitive type depends on the data type, while non-primitive types have all the same size.
- Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

Type Casting

- Type casting is when you assign a value of one primitive data type to another type.
- In Java, there are two types of casting:
  - **Widening Casting** (automatically) - converting a smaller type to a larger type size
    byte→short→char→int→long→float→double
  - **Narrowing Casting** (manually) - converting a larger type to a smaller size type
    double→float→long→int→char→short→byte

Example: **Widening Casting**

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Example: **Narrow Casting**

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

Variable

Variable

- The variable is the **basic unit of storage** in a Java program.
- The identifier is the name of the variable (or class name, …)
- The **valid identifier** has a series of characters consisting of **letters**, **digits**, underscores (_) and dollar signs (**$**) that **does not begin** with a **digit** and does **not contain spaces**.
- Some valid identifiers are Welcome1, $value, _value, m_inputField1 and button7.

```
int a, b, c; // declares three ints, a, b, and c.
int d = 3, e, f = 5; // declares three more ints, initializing // d and f.
byte z = 22; // initializes z.
double pi = 3.14159; // declares an approximation of pi.
char x = 'x'; // the variable x has the value 'x'.
```

[**Question**] Is the name **7button** a valid identifier?

Name Convention in Java

NAMING CONVENTIONS	APPLICATION	EXAMPLES
Lower Camel Case	variables and methods	firstName timeToFirstLoad indexNumber
Upper Camel Case	classes, interfaces, annotations, enums, records	TomcatServer RestController WriteOperation
Screaming Snake Case	constants	INTEREST_RATE MINIMUM_SALARY EXTRA_SAUCE
lower dot case	packages and property files	java.net.http java.management.rmi application.properties

Java Keywords

- The following list shows the **reserved words** in Java, which **may not be used as constant or variable** or any other identifier names.

abstarct	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Garbage Collection

- **Allocate and Deallocate memory**:
  - Java:
    - System responsibility:
      - Dynamically allocated memory
      - Deallocation is **done automatically** (system-level thread)
      - Checks for and frees memory no longer needed
  - C/C++:
    - Programmer's responsibility
      - Manually done (malloc & free)

Operators

Operators

All the **Java operators** can be divided into the following groups −

- **Arithmetic** Operators: * / % + -
- **Relational** Operators: < > <= >= == != **instanceof**
- **Bitwise** Operators: & ^ |
- **Logical** Operators: && || !
- **Assignment** Operators: = += -= *= /=
- **Ternary** operator: ? :

Math (Arithmetic) Operators

Operator	Description	Example	Output
+ (Addition)	Adds values A & B	A + B	32
- (Subtraction)	Subtracts B from A	A - B	-12
* (Multiplication)	Multiplies B by A	A * B	220
/ (Division)	Divides B by A	B / A	2.2
% (Modulus)	Divides left-hand by right-hand and return remainder	B % A	2

[**Question**] Can you guess what are A and B?

Relational Operators

```
int A = 5, B = 2;
```

Operator	Description	Example	Output
== (equal to)	Check if the values are equal.	A == B	false
!= (not equal to)	Check if the values are not equal.	A != B	true
> (greater than)	Check if the left value is greater than the right value.	A > B	true
< (less than)	Check if the left value is less than the right value.	A < B	false
>= (greater than or equal to)	Check if the left value is more than or equal to the right value	A >= B	true
<= (less than or equal to)	Check if the left value is less than or equal to the right value	A <= B	false
instance of	Check if the object is a particular type of (class type or interface type)	“hello” instance of String	true

Bitwise Operators

Operator	Description	Example	Output
		A = 3	00000011 (3)
		B = 5	00000101 (5)
& (bit-wise and)	Copies a bit if it exists in both.	A & B	00000001 (1)
| (bit-wise or)	Copies a bit if it exists in either.	A | B	00000111 (7)
^ (bitwise XOR)	Copies a bit if it is in one number but not both.	A ^ B	00000110 (6)
~ (bitwise compliment)	Flips all bits	~A	11111100 (-4)
<< (left shift)	Shift all bits to the left by a specified number.	A << 2	00001100 (12)

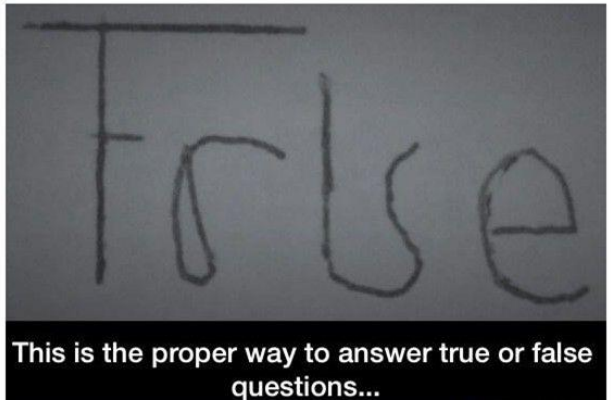
[Real Interview Question] Can you figure out how to know a number is even or odd using bitwise operators?

Logical Operators

```
boolean A = true, B = false;
```

Operator	Description	Example	Output
&& (logical and)	If both inputs are true then only that the condition is true.	A && B	false
|| (logical or)	If either of any input is true then the condition is true.	A || B	true
! (logical not)	Use to reverse the logical state of the input.	!(A && B)	true

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False



Assignment Operator

```
float A = 6, B = 3;
```

Operator	Description	Example	Output
= (assignment operator)	Assign values from the right side to the left side.	B = A + B	9.0
+= (Add AND assignment operator)	Is equivalent to B = B + A	B += A	9.0
-= (Subtract AND assignment operator)	Is equivalent to B = B - A	B -= A	-3.0
*= (Multiply AND assignment operator)	Is equivalent to B = B * A	B *= A	18.0
/= (Divide AND assignment operator)	Is equivalent to B = B / A	B /= A	0.5

Ternary Operator

- A special ternary (three-way) operator that can replace simple statement of if-then-else statements to have it in one line.
- expression1 ? expression2 : expression3
- For example,

```
int val1 = 10;

int val2 = 20;

int max = val1 >= val2 ? val1 : val2;
```

[Question] Can you guess what is the value of max?

Recap

This lecture, we have learnt about:

- Different Java Platforms
- Our choice of JDK
- The best IDE selections
- Create the first Java programs
- Compile and Run with commands or on IntelliJ
- Java data types:
  - Primitive
  - Non-primitive (reference types)
- Variable
- Operators

Thank you for your listening!