

- Java este un limbaj de programare **orientat-obiect**, conceput de către **James Gosling** la **Sun Microsystems**;
- Java ocupă un loc dominant în următoarele sectoare:
 - Aplicații Server;
 - Aplicații de telefonie mobilă;
 - Smart Devices – programe pentru casă intelligentă și dispozitive electronice.



- Este un limbaj actual și solicitat;
- A fost conceput cu gândul de a evita erorile de compilare și alocare de memorie;
- Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java;
- Programele Java pot fi lansate pe diferite dispozitiv: calculator, telefon, bancomat, toster, etc.



Dupa cum se stie, un orice program contine descrierea datelor si a algoritmului de prelucrare a acestor date pentru atingerea anumitor obiective. Se respecta astfel

$$\text{Program} = \text{Date} + \text{Algoritm}$$

In *programarea procedurala*, datele sunt descrise separat de prelucrari, astfel ca legatura dintre ele se realizeaza numai la nivel conceptual.

Pentru tipurile de date primitive s-a adoptat conceptul, conform caruia principiu de date se intlege o multime de *valori*, careia i se asociaza o multime de *operatii* care se pot efectua asupra valorilor respective.

În cazul *tipurilor de date derivate si a structurilor de date*, care sunt definite de catre programator, in limbajele procedurale aceasta legatura intre date si operatii nu se mai respecta, fiecare dintre ele fiind descrise in program separat

Programarea orientata pe obiecte este o metoda de implementare în care programele sunt organizate ca ansamblu de obiecte ce coopereaza intre ele, fiecare obiect reprezentand instanta unei clase, fiecare clasa apartine unei ierarhii in cadrul careia clasele sunt legate prin relatii.

Aceasta definitie cuprinde trei parti importante, si anume:

- obiectele si nu algoritmii sunt blocurile logice fundamentale;
- fiecare obiect este o instanta a unei clase. Clasa este o descriere a unei multimi de obiecte caracterizate prin structura si comportament similar;
- clasele sunt legate intre ele prin relatii.



In programarea *orientata pe obiecte* (in engleza: *OOP - Object Oriented Programming*) , legatura stransa dintre date si operatiile efectuate asupra lor se extinde si asupra structurilor de date.

Fiecare **obiect** este o *structura de date*, asociata cu o colectie de *metode* (functii sau proceduri) prin intermediul carora se manipuleaza aceste date. Obiectele care au aceeasi structura si aceleasi metode se grupeaza intr-o *clasa*.

Clasa este, deci, o extindere a conceptului de tip de date, in care multimea de valori este o multime de obiecte, iar multimea operatiilor este o multime de metode. "Programul" este, in POO, un ansamblu de clase si obiecte care comunica intre ele prin mesaje.

Fiecare obiect se caracterizeaza prin *stare si comportament*.

Starea obiectului este caracterizata prin valorile pe care le au, la un moment dat, datele continute de acesta. Daca se modifica una sau mai multe astfel de valori, se modifica si starea obiectului.

Comportamentul obiectului se manifesta in modul in care obiectul raspunde la primirea unui mesaj. Acest raspuns poate depinde atat de mesajul primit, cat si de starea in care se gaseste obiectul in momentul primirii acestuia.

Tratarea mesajului se face prin aplicarea *metodelor* obiectului respectiv. Din punctul de vedere al programarii procedurale, *metoda* poate fi o *functie* sau o *procedura*. Deosebirea dintre ele este ca *functia* este invocata (apelata) pentru a obtine *valoarea ei* (numita si *valoare intoarsa*), in timp ce *procedura* nu intoarce o *valoare*, ci se executa pentru a obtine un *efect lateral* (de exemplu citirea unor date de la tastatura, afisarea unor date pe ecran etc.)

In majoritatea surselor bibliografice asupra POO, se considera drept principale caracteristici ale obiectelor *incapsularea*, *mostenirea* si *polimorfismul*. La acestea se mai pot adauga si alte caracteristici importante, cum sunt *identitatea*, *agregarea* si *clasificarea*.

Identitatea (engleza: *Identity*) se refera la faptul ca datele sunt grupate in entitati discrete, numite *obiecte*.

Fiecare obiect are propria lui *identitate*, astfel ca doua obiecte sunt considerate distincte, chiar daca atributele lor (cum ar fi numele, culoarea etc.), sunt identice. Pentru a face aceasta distinctie, obiectul este indicat printr-o *referinta unica*. Modul in care este reprezentata aceasta referinta poate sa difere in diverse limbaje de programare (de ex. adresa de memorie, indice de tablou etc.), important insa este ca fiecare obiect are o singura referinta si nu exista doua obiecte distincte cu aceeasi referinta.

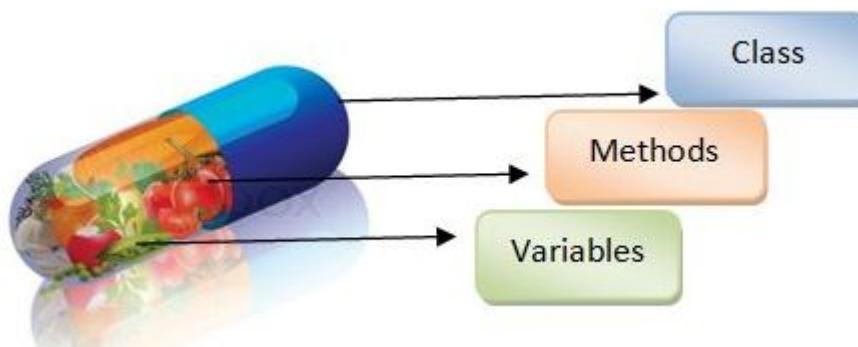
Agregarea (engleza: *aggregation*) este proprietatea obiectelor de a putea incorpora alte obiecte. Așa dar, "datele" continute într-un obiect pot fi nu numai date primitive, ci și obiecte. Se pot astfel crea obiecte cu structuri din ce în ce mai complexe.

Clasificarea (engleza: *classification*) este proprietatea obiectelor care au aceeași *structura de date și același comportament* (aceeași metode) de a putea fi grupate într-o *clasa*. Clasa este o *abstractizare*, care conține acele proprietăți ale obiectelor, care sunt importante într-o aplicație sau într-o categorie de aplicații, și le ignora pe celelalte. De exemplu, în aplicații privind situația profesională a studentilor, clasa *Student* conține astfel de atribute ca numele și prenumele studentului, facultatea, anul de studii, examenele promovate și notele obținute, dar ignora atribute ca înaltimea, culoarea ochilor sau a parului, greutatea etc., care nu sunt necesare în aplicația respectivă. Spre deosebire de *agregare*, care este o relație între obiecte, *clasificarea* este o *relație între concepte* reprezentate prin *clase*.

Incapsularea (engleza: *encapsulation*) este proprietatea obiectelor de a-si ascunde o parte din date si metode. Din exteriorul obiectului sunt accesibile ("vizibile") numai datele si metodele *publice*.

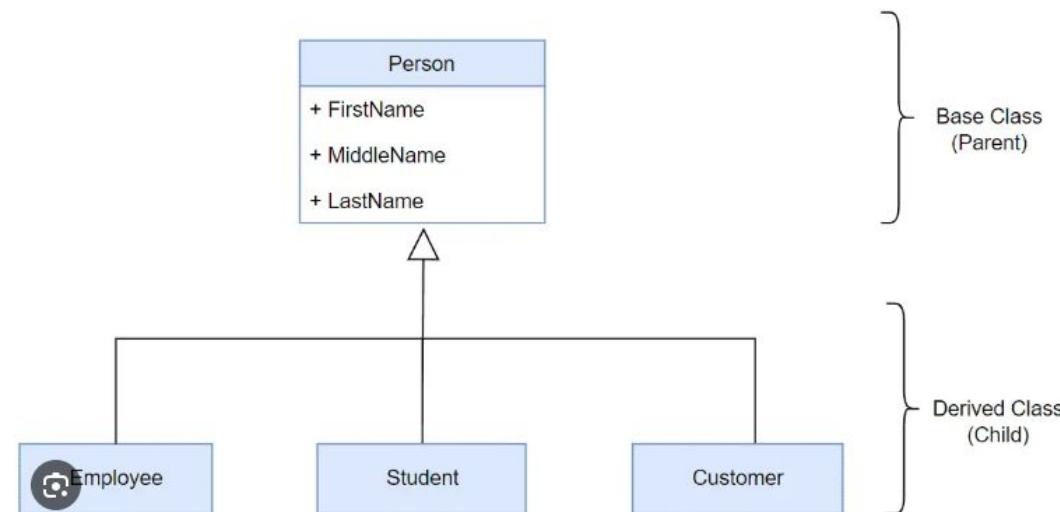
Starea obiectului depinde atat de datele publice, cat si de cele encapsulate. Metodele publice ale obiectului au acces la datele si metodele encapsulate (ascunse) ale acestuia.

In consecinta, starea obiectului poate fi modificata atat prin modificarea directa, din exterior, a valorilor variabilelor publice, fie prin utilizarea unor metode publice care modifica valorile variabilelor encapsulate. In mod similar, valorile variabilelor encapsulate pot fi obtinute numai utilizand metode publice ale obiectului respectiv.

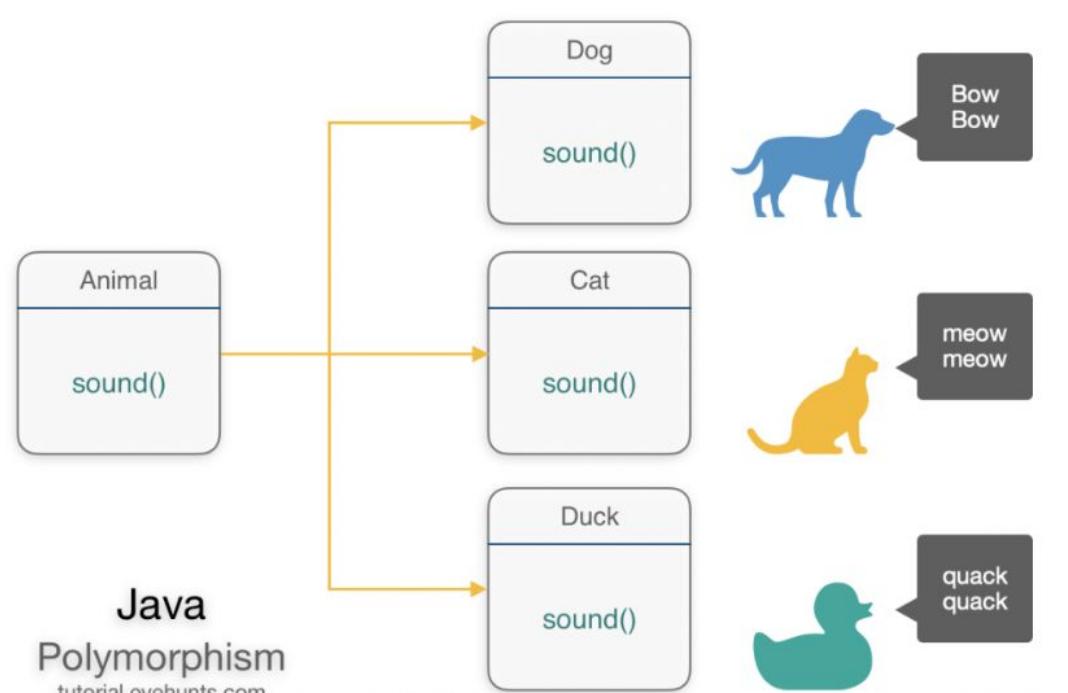


Mostenirea (engleza: *inheritance*) este proprietatea unei clase de a contine toate atributele (variabilele) si metodele superclasei sale. In consecinta, trecerea de la clasa la subclasa se face prin adaugarea de atrbute si/sau de metode. De exemplu, clasa *Student* si clasa *Lector* au ambele toate atrbutele clasei *Om*, dar fiecare din acestea are si atrbute specifice.

In general, in programarea orientata pe obiecte, mostenirea poate fi simpla sau multipla. In cazul *mostenirii simple* fiecare clasa are cel mult o superclasa, in cazul *mostenirii multiple* o clasa poate avea mai multe superclase.



Polimorfismul (engleza: *polymorphism*) permite ca aceeasi operatie sa se realizeze in mod diferit in clase diferite. Sa consideram, de exemplu, ca in clasa *Figura_geometrica* exista metoda *arie()*, care calculeaza aria figurii respective. Clasele *Cerc*, *Triunghi*, *Patrat* sunt subclase ale clasei *Figuri_geometrice* si vor mosteni, deci, de la aceasta metoda *arie()*. Este insa evident ca aria cercului se calculeaza in alt mod decat aria patratului sau cea a triunghiului. Pentru fiecare din instancele acestor clase, la calcularea ariei se va aplica metoda specifica clasei sale.



Supraîncărcarea și supradefinirea metodelor sunt două concepte extrem de utile ale programării orientate obiect, cunoscute și sub denumirea de *polimorfism*, și se referă la:

- *supraîncarcarea (overloading)* : în cadrul unei clase pot exista metode cu același nume cu condiția ca signaturile lor să fie diferite (lista de argumente primite să difere fie prin numărul argumentelor, fie prin tipul lor) astfel încât la apelul funcției cu acel nume să se poată stabili în mod unic care dintre ele se execută.

supradefinirea (overriding): o subclasă poate rescrie o metodă a clasei părinte prin implementarea unei metode cu același nume și aceeași signatură ca ale superclasei.

Una dintre notatiile recunoscute la ora actuala ca un standard in cadrul tehnologiei obiectuale este **UML** (Unified Modelling Language)

O reprezentare grafica in notatie UML se numeste **diagrama**. O diagrama arata ca un **graf** in care nodurile pot fi clase, obiecte, stari ale unui obiect, iar arcele reprezinta relatii intre noduri. Pentru acelasi program se pot utiliza mai multe tipuri de diagrame, care surprind mai multe aspecte ale programului.

Pentru a reda structura de clase a unui program vom utiliza asa-numitele **diagrame de clase** (class diagram). Intr-o asemenea diagrama nodurile sunt clasele, iar arcele sunt relatiile stabilite intre clase. In UML o clasa se reprezinta astfel:

Nume_clasa

Campuri

Metode

Exceptand compartimentul cu numele clasei, care trebuie completat obligatoriu, celelalte compartimente pot sa ramana necomplete, daca intr-un anumit stadiu de proiectare nu ne intereseaza acele elemente. In dreptul numelor de campuri si de metode pot sa apară anumite simboluri reprezentand modificatorii de acces:

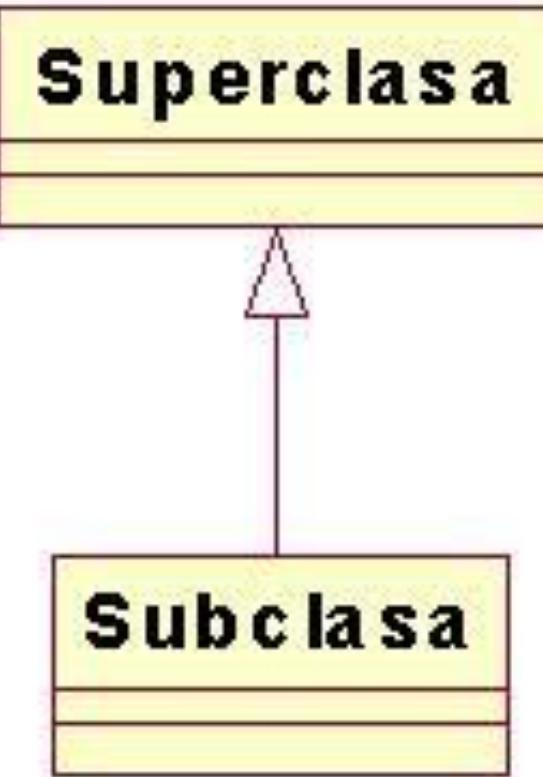


Intre clase pot sa apară mai multe tipuri de relații.

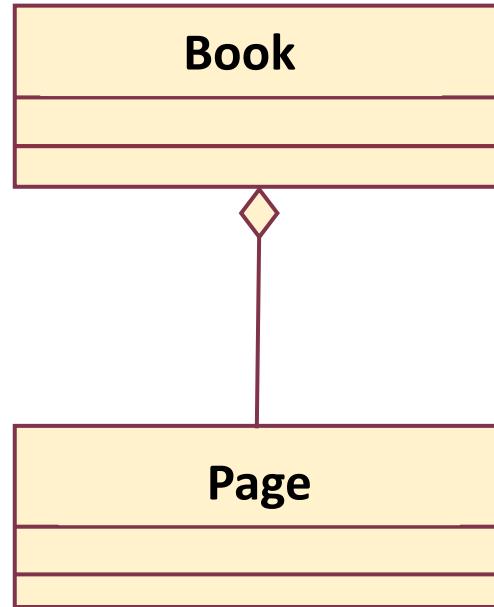
- intre o clasa server și un client al ei există o **relație de utilizare** . Relația de utilizare reprezinta de fapt un caz particular al unei relații mai generale numita **asociere**. Ca să fim mai exacti, relația de asociere se stabilește între obiecte, dar într-o diagramă putem figura relația respectiva ca o legătura între clasele la care aparțin acele obiecte.



Între o subclasa si o superclasa exista o **relatie de mostenire**
sau de **generalizare/specializare**



Între o subclasa si o superclasa exista o **relatie de agregare- apartenență** (pagina aparține carte)



Relația de **Metaclasa** (regula de structura, creare si prelucrare a claselor)

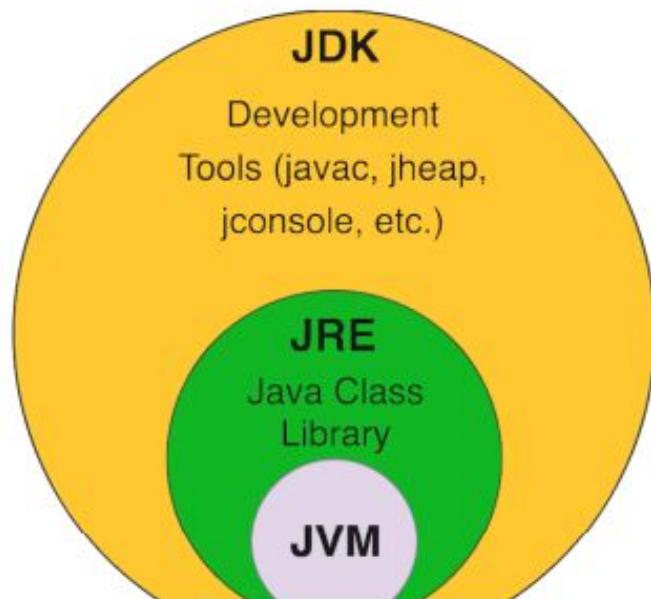
JRE vs. JDK

Java Runtime Environment (JRE)

- Required to run Java apps
- End-users normally require only the JRE

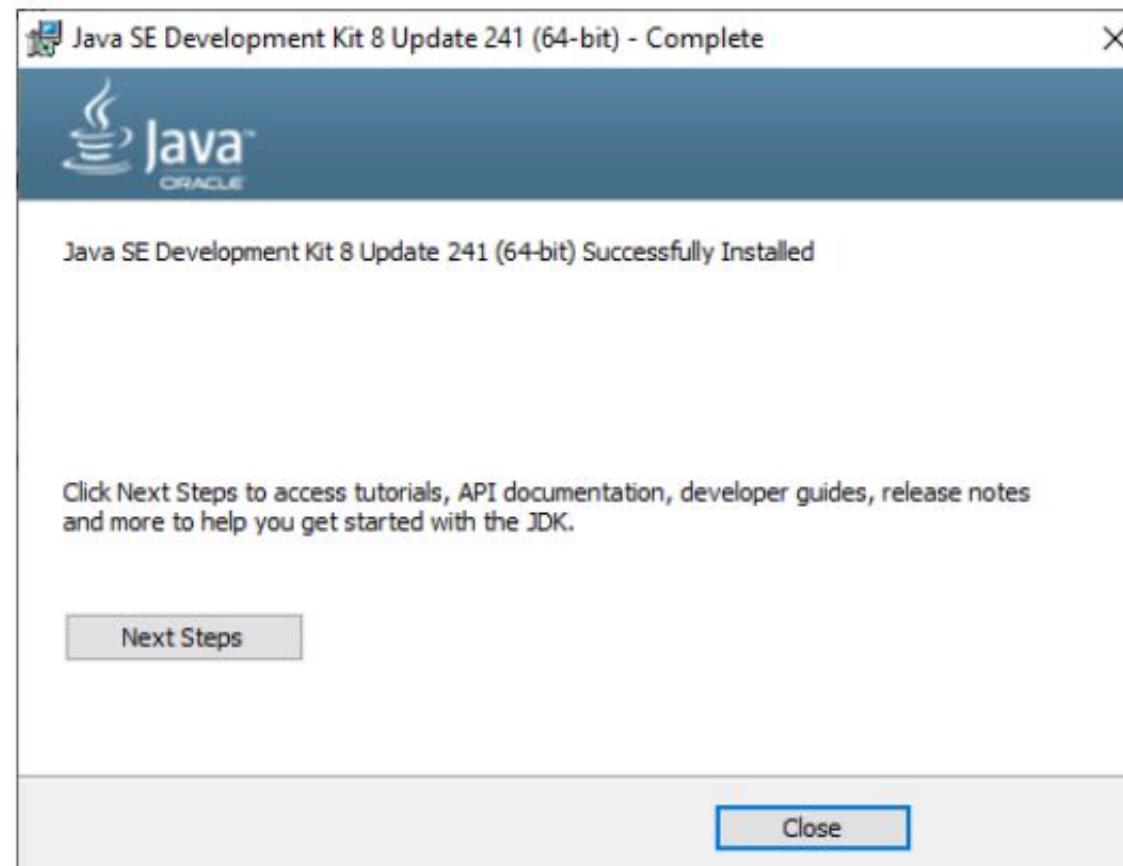
Java Development Kit (JDK)

- Provides tools required to create Java apps
- Developers normally require the JDK
- JDK installation includes JRE



Exercise #1 Configuring the JDK

Click the **Close** button.



Exercise #2 Configuring the IDE

1. Download [IntelliJ IDEA Community](#).
2. Double click on the installer.
3. Follow the screens that appear.

Using the IDE

A Java **Integrated Development Environment** (IDE) is a type of software that makes it easier to develop Java applications.

An **IDE** provides:

- Syntax checking
- Various automation features
- Runtime environment for testing
- Organizes all Java resources and environment settings into a Project
- Projects contain packages
- Packages contain files, such as `.java`

Using the IDE

Project
Navigator

The screenshot shows the IntelliJ IDEA interface with the following components labeled:

- Project Navigator**: Located on the left side, it displays the project structure. A callout arrow points from the text "Project Navigator" to the "Project" tool window.
- Class Navigator**: Located at the bottom left, it shows the class hierarchy and methods for the selected class. A callout arrow points from the text "Class Navigator" to the "Structure" tool window.
- Code editor**: The main central area where the Java code is written. A callout arrow points from the text "Code editor" to the code editor window.

The code editor displays the following Java code for the `Lamp` class:

```
package edu.tekwill.java.marathon.lamp;

public class Lamp {

    boolean status;
    int luminosity;

    public void turnOn() {
        status = true;
        luminosity = 50;
    }

    public void turnOff() {
        status = false;
        luminosity = 0;
    }

    public void increaseLuminosity() {
        if (status && luminosity < 100) {
            luminosity += 25;
        }
    }

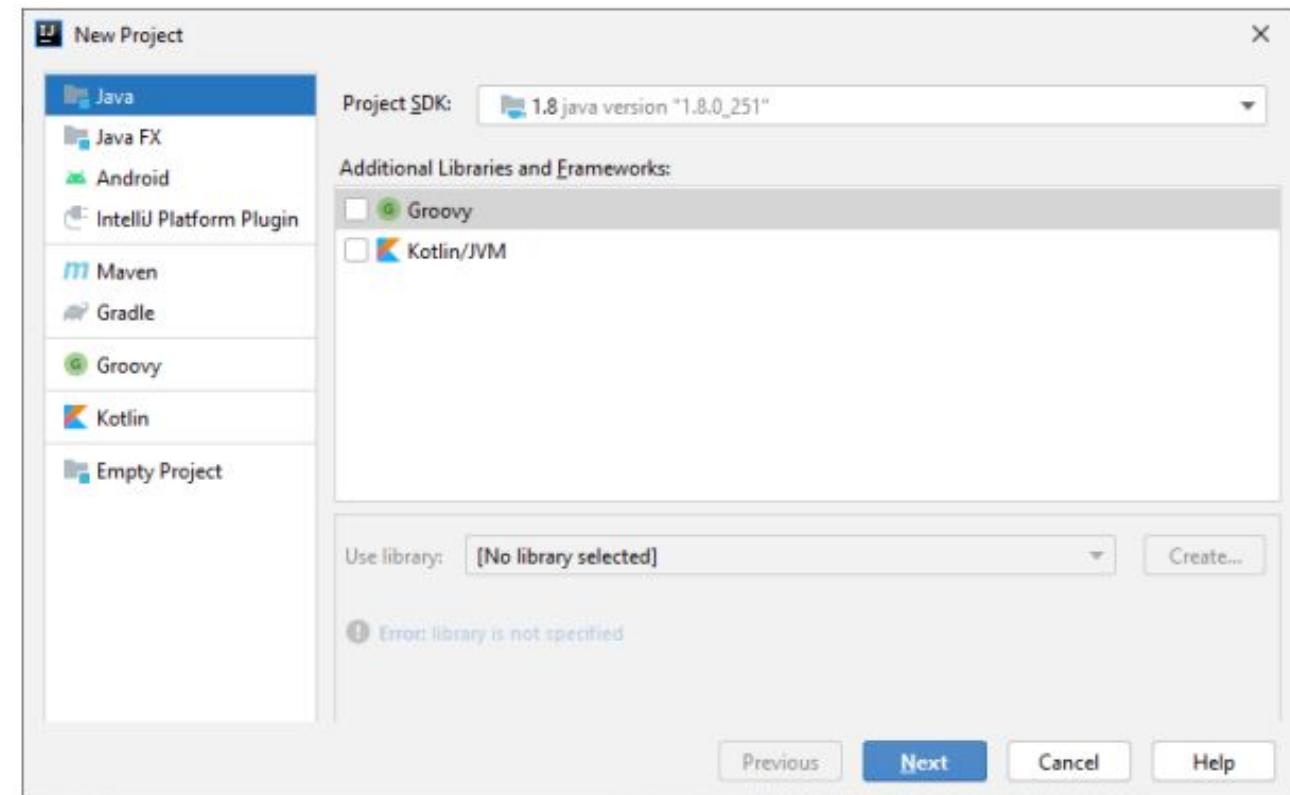
    public void decreaseLuminosity() {
        if (status && luminosity > 25) {
            luminosity -= 25;
        }
    }

    public String toString() {
        String onOffMsg = status ? "on" : "off";
        return String.format("Lamp is %s, luminosity=%s", onOffMsg, luminosity);
    }
}
```

Class
Navigator

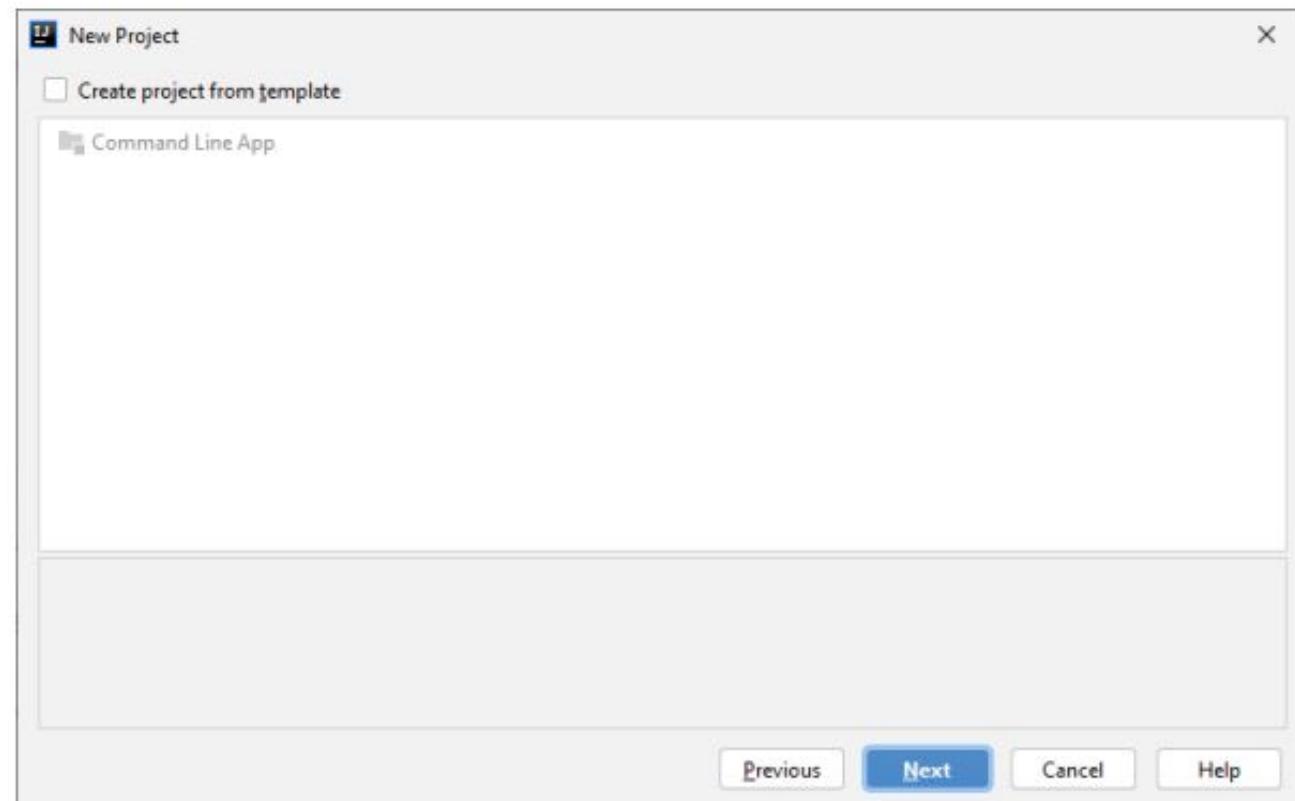
Creating a Java Project

1. Select File > New > Project.
2. Select Java.
3. Click Next.



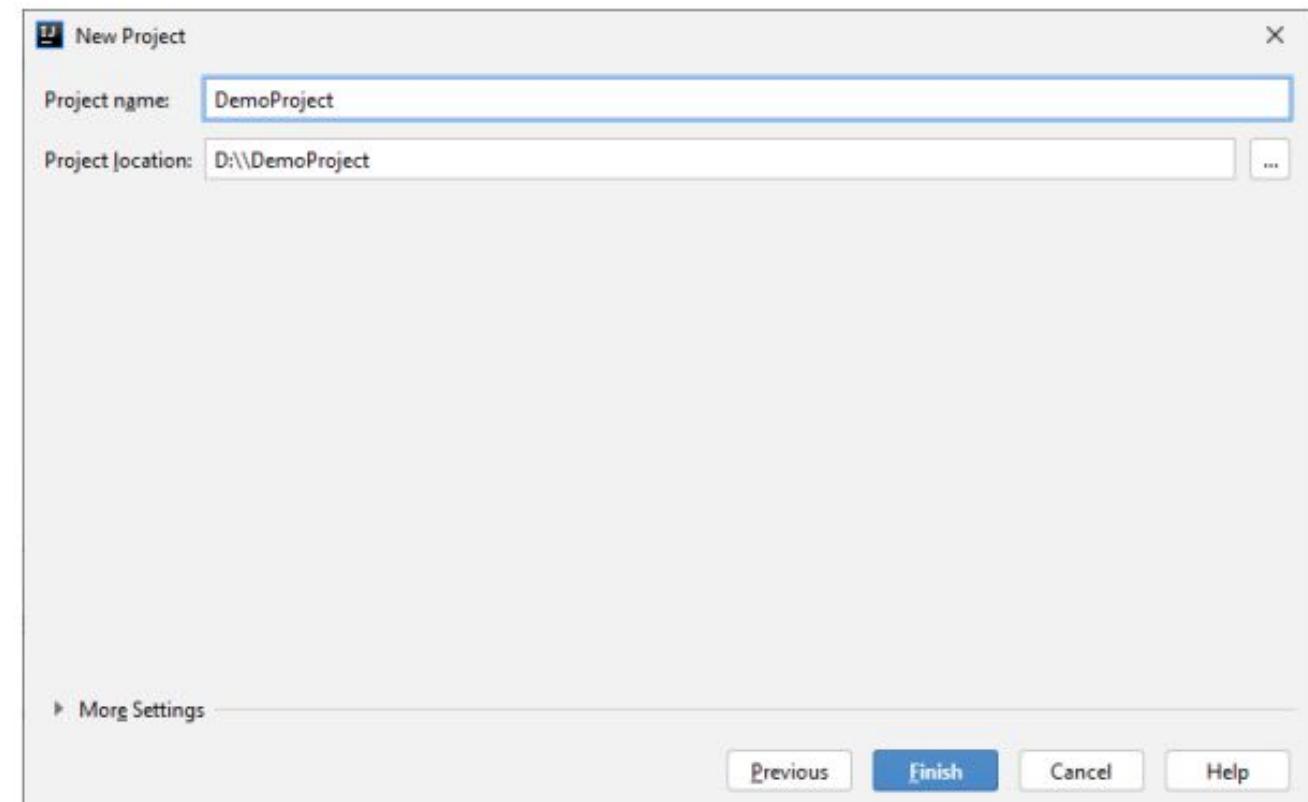
Creating a Java Project

4. Click Next again.



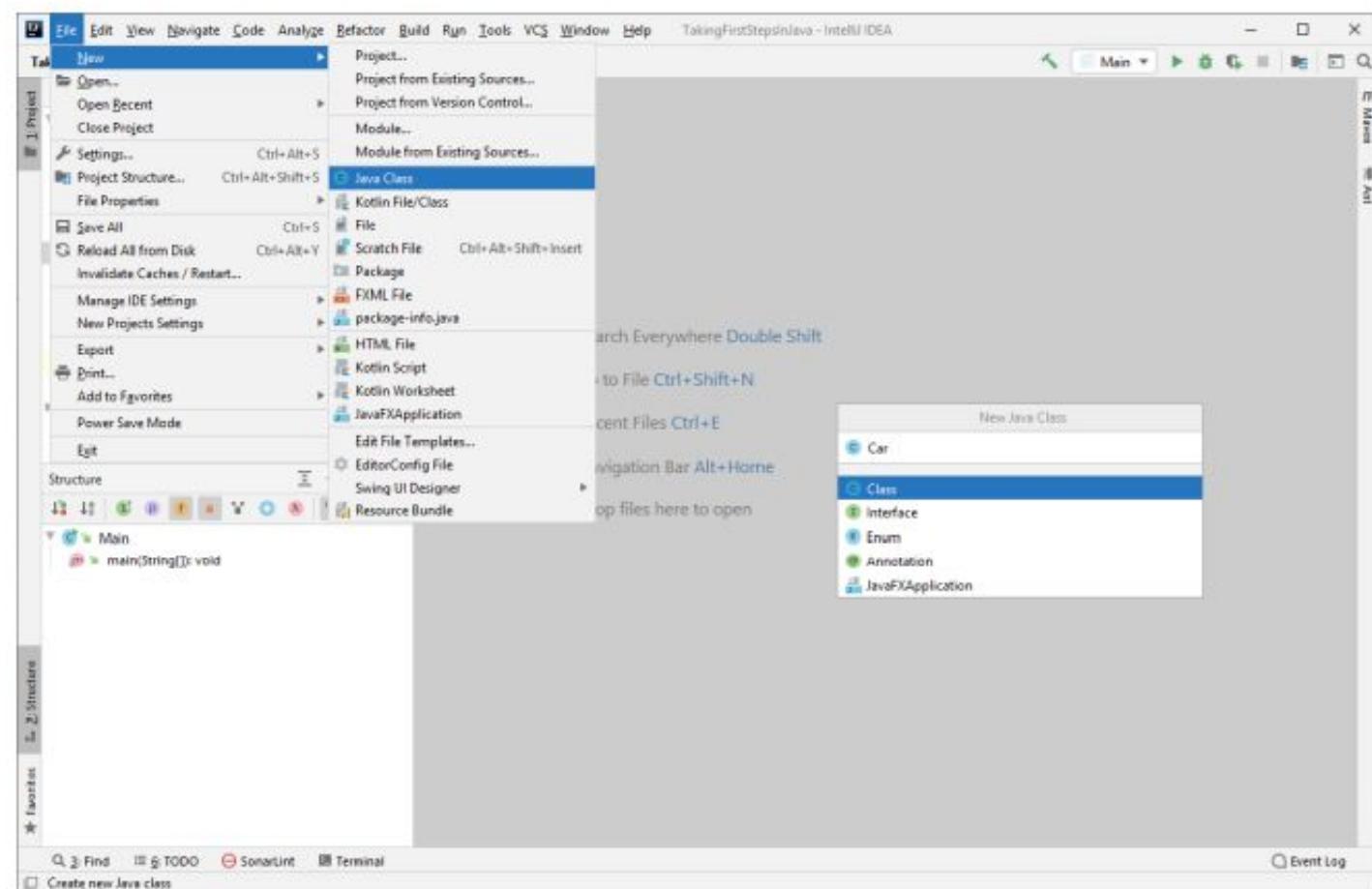
Creating a Java Project

5. Set the project name and the location on the disk where you want it to be saved.
6. Click **Finish**.



Creating a Java Class

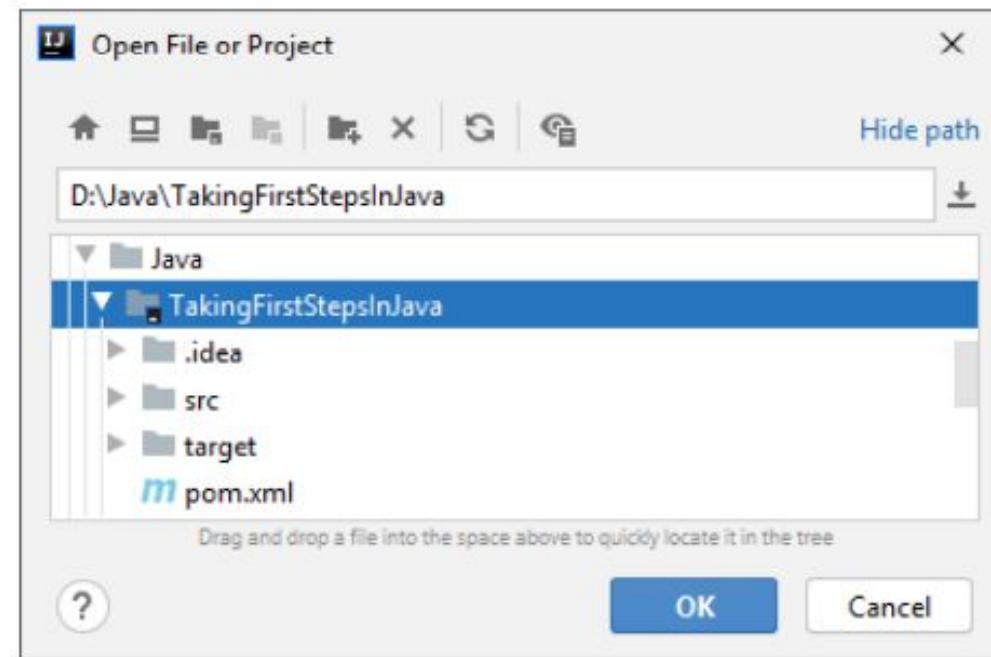
1. Select File > New > Java Class.
2. Define the name of the class.
3. Press Enter.



Opening an Existing Java Project

If you ever need to open an existing project, perform the following steps:

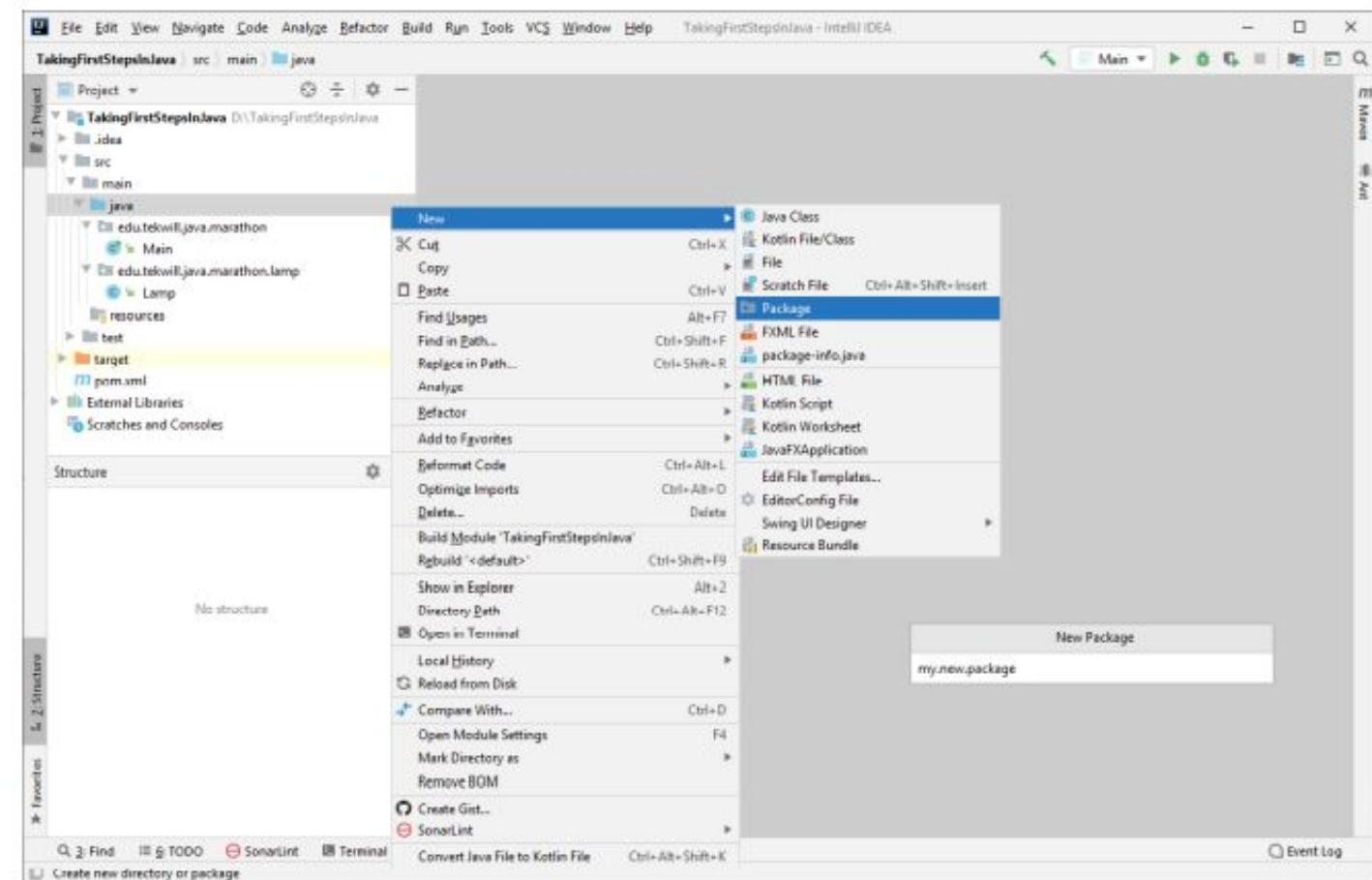
1. Select **File > Open**.
2. Navigate to the directory that contains your projects.
3. Select the project file you want.
4. Click **Ok**.



Creating a New Java Package

If you ever need to create a new package, perform the following steps:

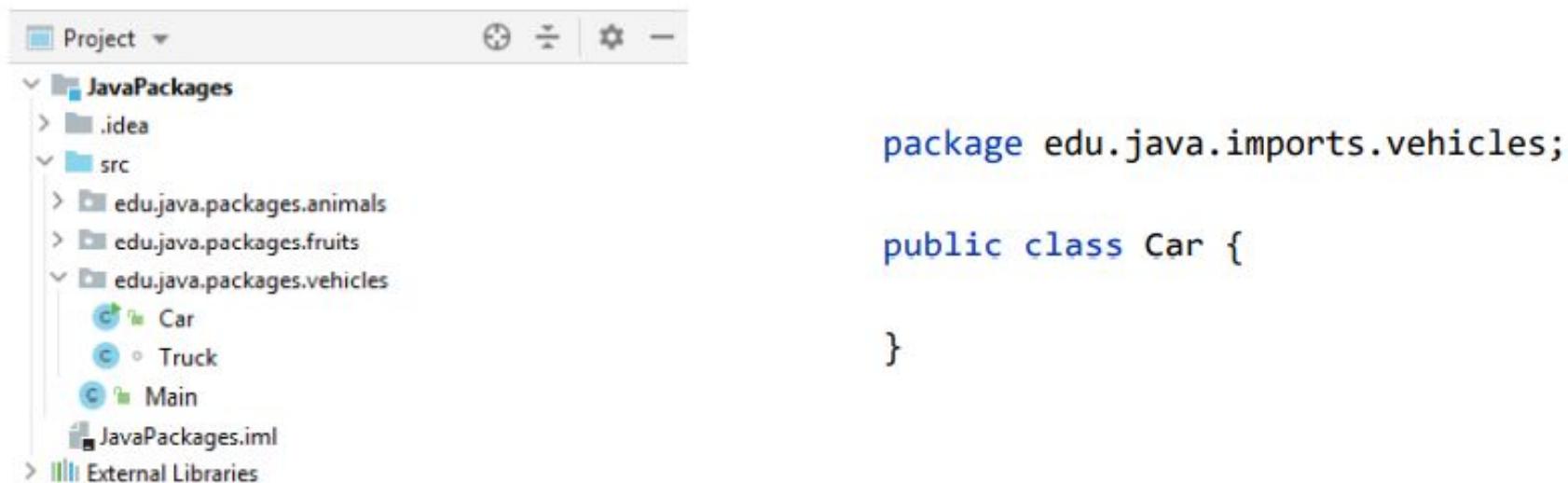
1. Right-click the **java** folder.
2. Select **New > Package**.
3. Define the name of the package.
4. Press **Enter**.



Definition of a Class in a Java Code File

When you define a *public* class in a Java source file, the name of the class and Java source file must match.

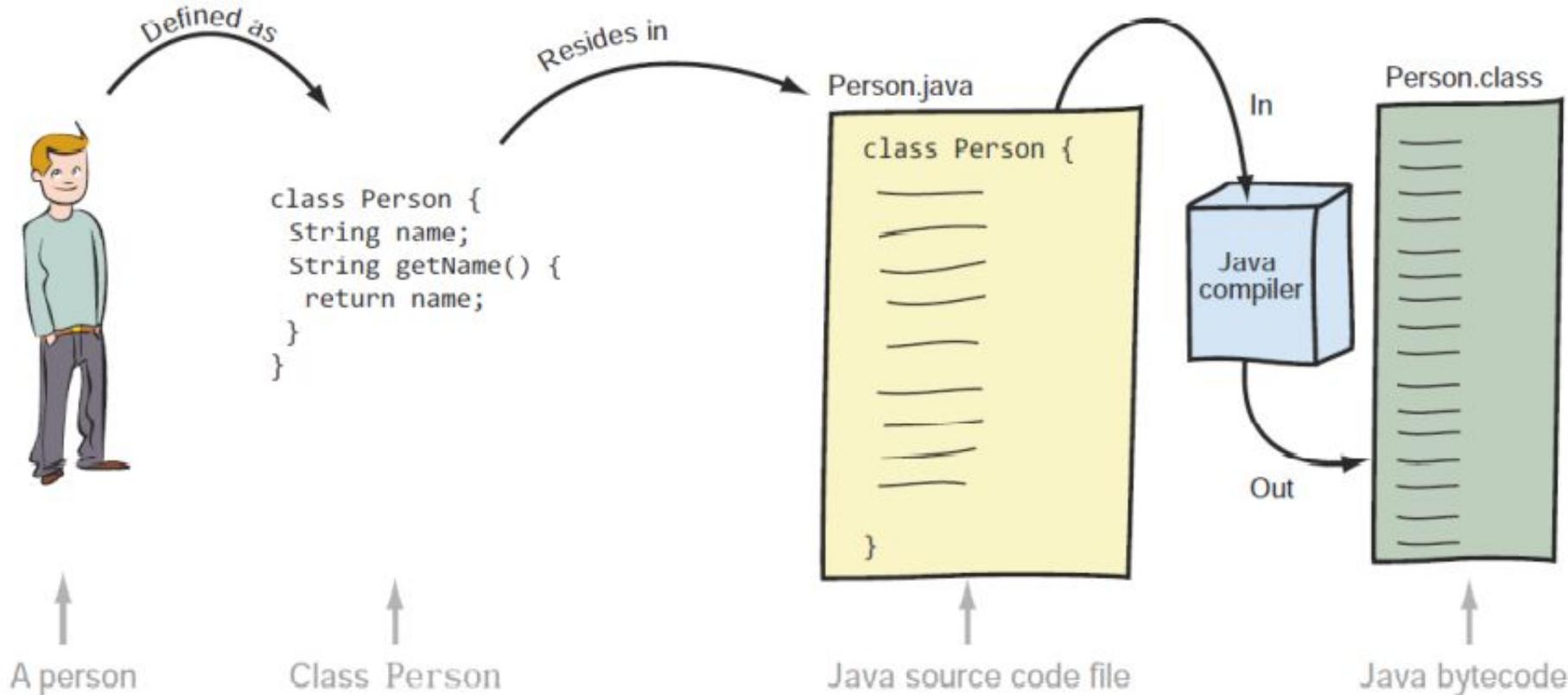
A source code file can't define more than one public class. If you try to do so, your code won't compile.



```
package edu.java.imports.vehicles;

public class Car {
```

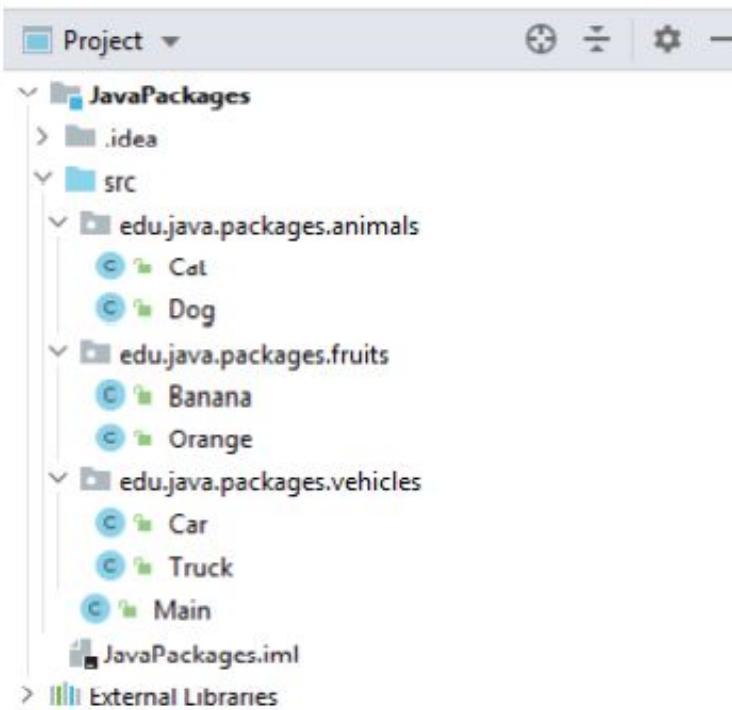
Java Source Code File vs Java Bytecode File



Java Packages

All Java classes are part of a package.

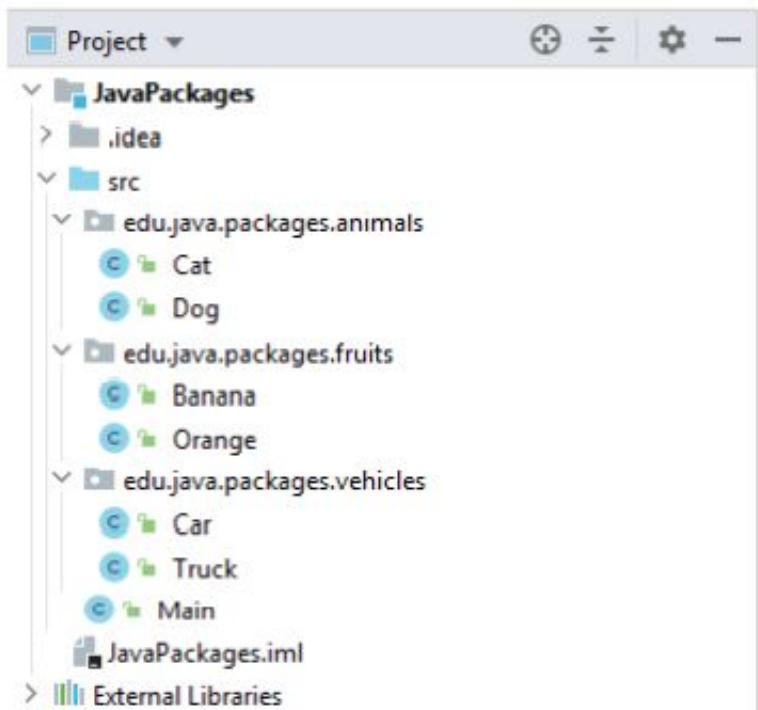
A Java class can be explicitly defined in a named package; otherwise, it becomes part of a default package, which doesn't have a name.



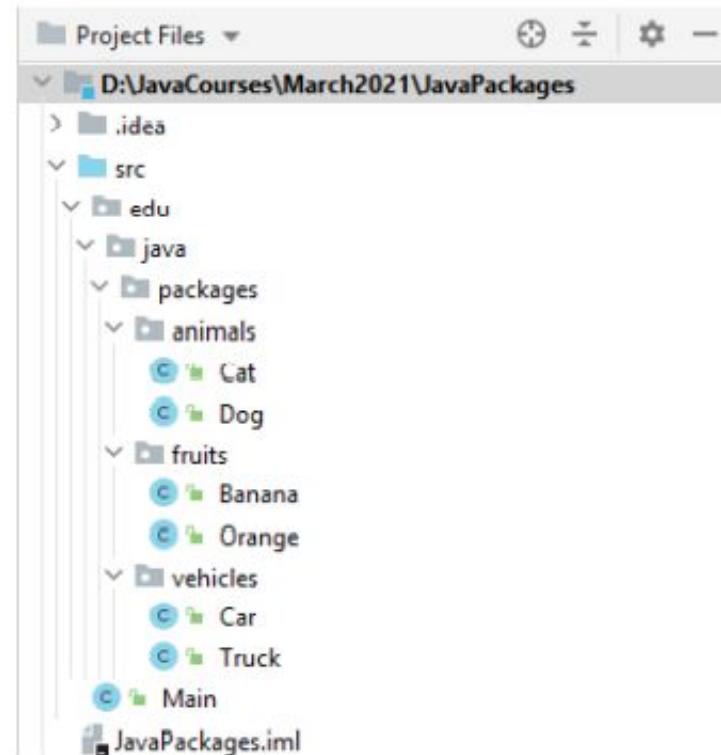
Clasele în cadrul unei aplicații complexe nu sunt plasate independent. Sunt incluse în pachete (~folder)

- conțin elemente omogene (clase, interfețe, subpachete, etc.);
- categorizează concepte (conform criteriilor: funcționalitate, tip, destinație etc.);
- restricționează accesul;
- evită conflicte la nivel de nume.

Java Packages



Classes structured
in packages in IntelliJ

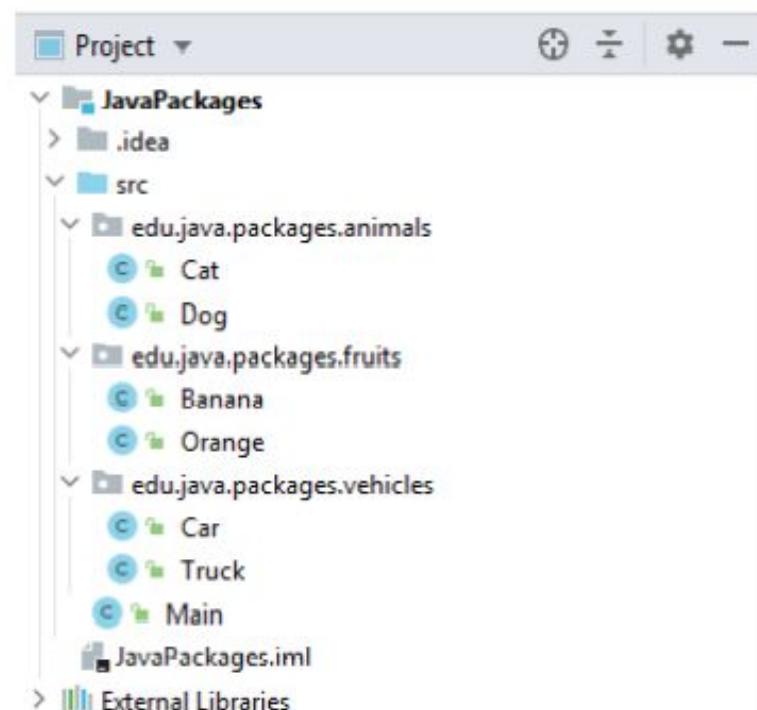


Classes structured
in packages in file system

Java Packages

```
package edu.java.packages.animals;

/**
 * @author nsirbu
 */
class Dog {
```



Java Packages

Example:

com.oracle.javacert.associate

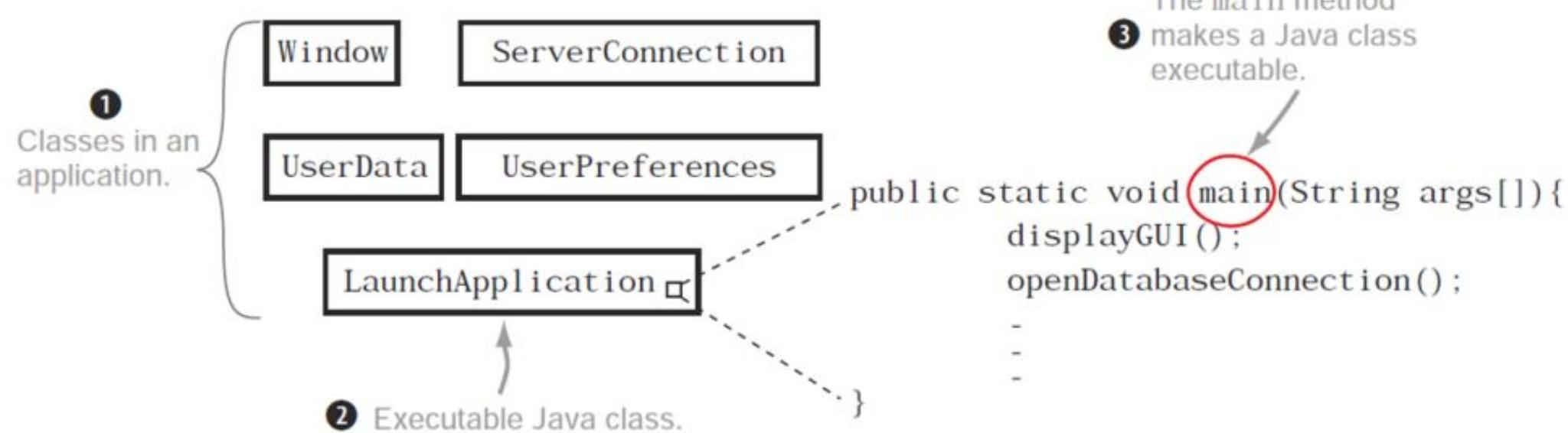
Package or subpackage name	Its meaning
com	Commercial. A couple of the commonly used three-letter package abbreviations are <ul style="list-style-type: none"><li data-bbox="942 889 1556 938">■ gov—for government bodies<li data-bbox="942 947 1633 996">■ edu—for educational institutions
oracle	Name of the organization
javacert	Further categorization of the project at Oracle
associate	Further subcategorization of Java certification

Java Packages

Here are a few of important rules about packages:

- Per Java naming conventions, package names should all be in lowercase.
- The package and subpackage names are separated using a dot.
- For classes and interfaces defined in a package, the package statement is the first statement in a Java source file. The exception is that comments can appear before or after a package statement.
- There can be a maximum of one package statement per Java source code file.

Executable Java Classes vs Non-Executable Java Classes



Class `LaunchApplication` is an executable Java class, but the rest of the classes - `Window`, `UserData`, `ServerConnection`, and `UserPreferences` - aren't.

The main Method



It is a special method that the JVM recognizes as the starting point of any Java program.

This `main` method should comply with the following rules:

- The method must be marked as a `public` method.
- The method must be marked as a `static` method.
- The name of the method must be `main`.
- The return type of this method must be `void`.
- The method must accept a method argument of a `String array` or a variable argument of type `String`.

A `main` Class Example

```
class FirstApp {  
    public static void main(String[] args) {  
        System.out.println("Hello world!");  
    }  
}
```

The access modifier must be `public`.

The nonaccess modifier must be `static`.

The method should not return a value; its return type must be `void`.

The name of the method must be `main`.

The method must accept an array or varargs of type `String`. The name of the method parameter can be any valid identifier name.

Exercise #3 Creating the `main` method

1. Open a text editor, Notepad++ for example.
2. Create a file.
3. Create a new class inside that file.
4. Add the method `main` to the body of the class you created:

```
public static void main(String[] args) { ... }
```

1. In the body of the `main` method, use a `System.out.println` to print the following message:
“Welcome to the `main` method of my application!”.
2. Compile and Run your class (see next slide).

Exercise #4 Running a Java program from CMD

Running a class that has the `main` method from command line, requires you to do the following operations:

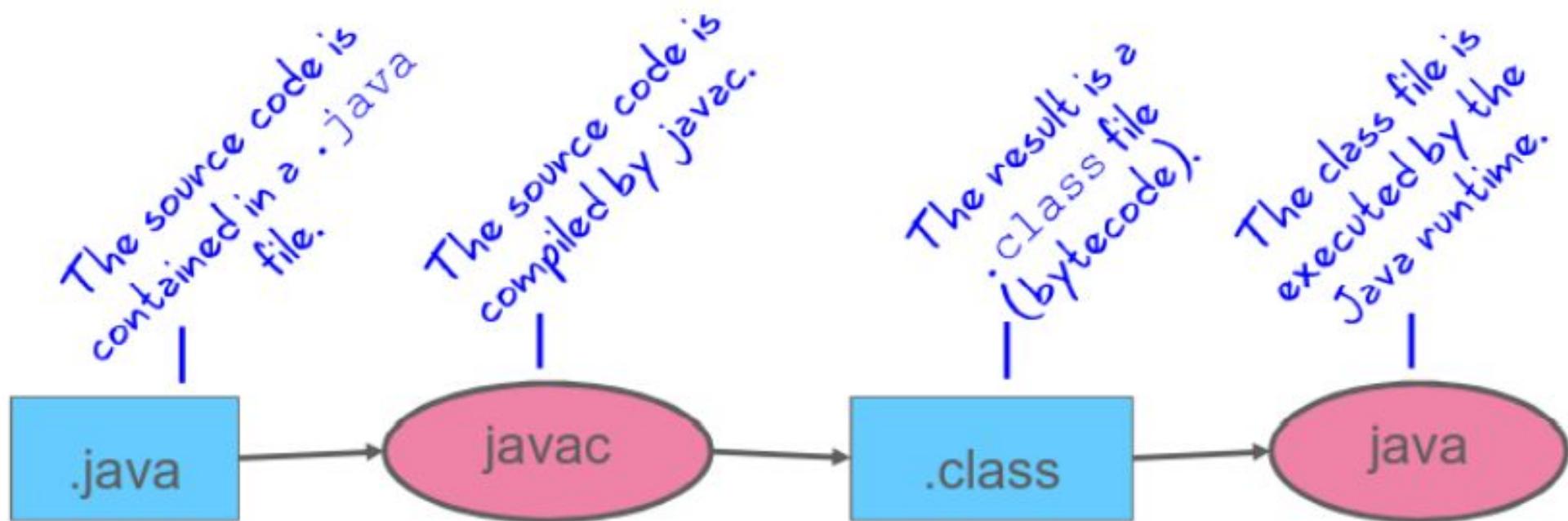
1. First of all, you need to use the `javac` command to compile the class:

```
javac HelloWorld.java
```

1. Then you can use the `java` command to run it:

```
java HelloWorld
```

Compiling and Running a Java Program



Avoiding Syntax Errors

The IDE will tell you if you have done something wrong.

Common errors include:

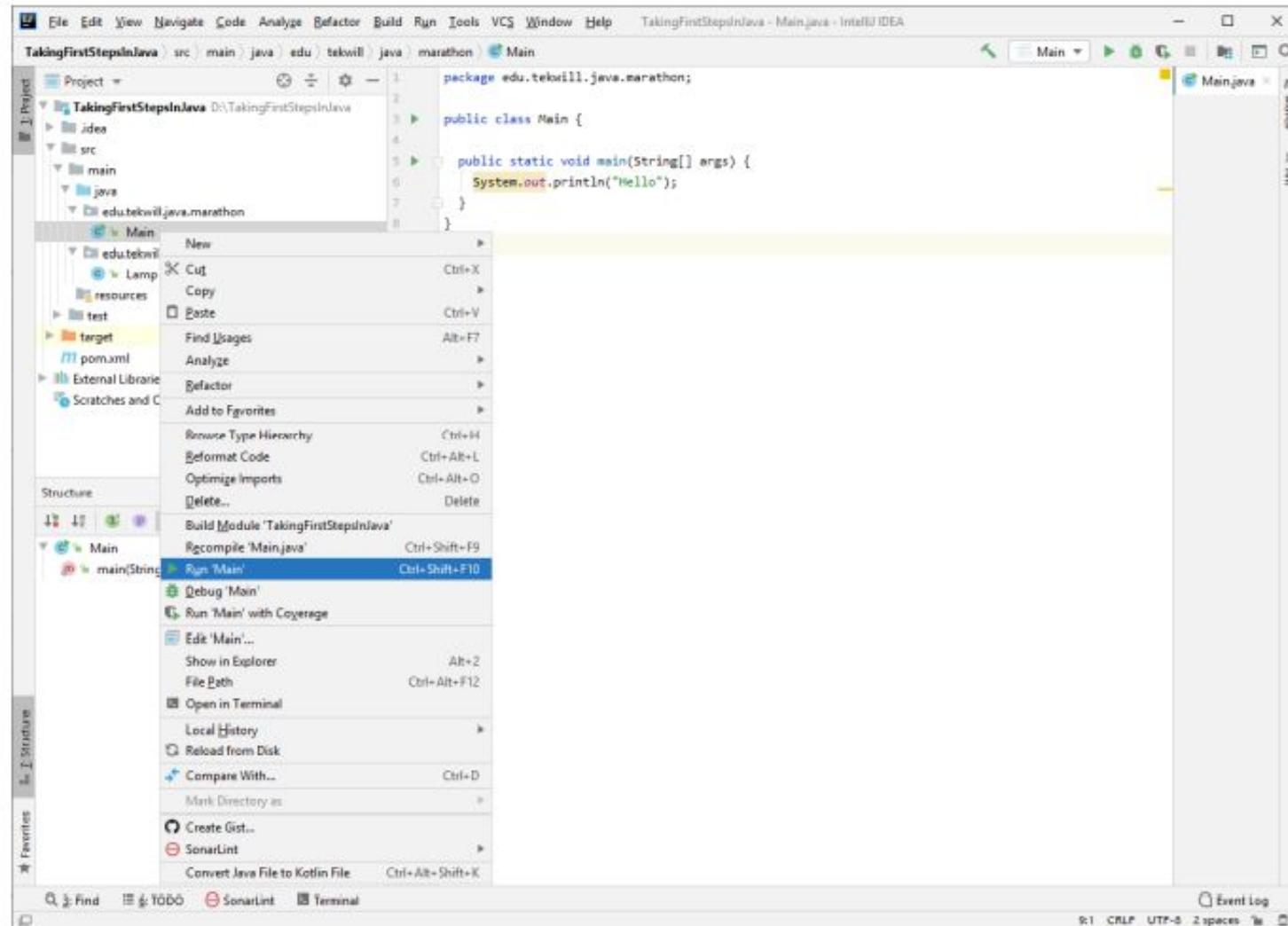
- Unrecognized words (case-sensitivity errors)
- Missing close quotation mark - “
- Unmatched brace - { }
- Missing semicolon - ;

The screenshot shows a Java code editor with the following code:

```
1 package edu.tekwill.java.marathon;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         System.out.println("Hello")
7     }
8 }
```

A red box highlights the closing brace of the main method. A tooltip window appears over the error, containing the text "';' expected". The status bar at the bottom of the IDE shows the message "Main.java:1: error: ';' expected".

Compiling and Running Using IntelliJ IDEA



Exercise #6 Creating the `main` method

1. Open IntelliJ IDEA.
2. Create a new Java project.
3. Create a new class in the default package.
4. In the code editor add the method `main` to the class you created:

```
public static void main(String[] args) { ... }
```

1. In the body of the `main` method, use a `System.out.println` to print the following message:
“Welcome to the `main` method of my application!”.
2. Right click on your class. Choose **Run ‘Main’** from the context menu.

Regulile de care se ține cont la declararea variabilelor în Java:

- numele variabilei trebuie să înceapă cu o literă, linie de subliniere (_) sau cu simbolul dolar (\$);
- numele de variabila NU poate începe cu o cifră;
- după primul caracter se pot folosi cifre
- numele de variabila NU poate fi un cuvânt Java rezervat ([Java keywords](#));
- pot fi declarate mai multe variabile în același timp.

Properties of valid identifiers	Properties of invalid identifiers
Unlimited length	Same spelling as a Java reserved word or keyword
Starts with a letter (a–z, upper- or lowercase), a currency sign, or an underscore	Uses special characters: !, @, #, %, ^, &, *, (,), ', :, ;, [, /, \, }
Can use a digit (not at the starting position)	Starts with a Java digit (0–9)
Can use an underscore (at any position)	
Can use a currency sign (at any position): ₩, \$, £, ¢, ¥, and others	

Examples of valid identifiers	Examples of invalid identifiers
customerValueObject	7world (identifier can't start with a digit)
\$rate, fValue, _sine	%value (identifier can't use special char %)
happy2Help, nullValue	Digital!, books@manning (identifier can't use special char ! or @)
Constant	null, true, false, goto (identifier can't have the same name as a Java keyword or reserved word)

Data type	Size	Range of values
byte	8 bits	-128 to 127, inclusive
short	16 bits	-32,768 to 32,767, inclusive
int	32 bits	-2,147,483,648 to 2,147,483,647, inclusive
long	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807, inclusive

```
long baseDecimal = 100_267_760;           ▷ int baseDecimal = 267;  
long octVal = 04_13;                     int octVal = 0413;  
long hexVal = 0x10_BA_75;                 int hexVal = 0x10B;  
long binVal = 0b1_0000_10_11;              int binVal = 0b100001011;
```

char	Character represented in 16-bit Unicode '\u0000' to '\uFFFF'. Can be treated as integer in the range of [0, 65535] in arithmetic operations. (Unlike C/C++, which uses 8-bit ASCII code.)
------	---

```
char c1 = 122;           ← Assign z to c1
```

```
char c2 = '\u0122';
System.out.println("c1 = " + c1);          c1 = z
System.out.println("c2 = " + c2);          c2 = G
```

boolean	Binary that takes a literal value of either true or false. The size of boolean is not defined in the Java specification, but requires at least one bit. booleans are used in <i>test</i> in decision and loop, not applicable for arithmetic operations. (Unlike C/C++, which uses integer 0 for false, and non-zero for true.)
---------	--

Data type	Size	Range of values
float	32 bits	$+/-1.4E-45$ to $+/-3.4028235E+38$, $+/-\infty$, $+/-0$, NaN
double	64 bits	$+/-4.9E-324$ to $+/-1.7976931348623157E+308$, $+/-\infty$, $+/-0$, NaN

```
float average = 20.129F;  
float orbit = 1765.65f;  
double inclination = 120.1762;
```

Exemple de initializări corecte de variabile cu tipuri de date primitive:

```
int value1;  
int value2;  
value1 = value2 = 17;  
int valueB8 = 021;  
int valueB16 = 0x11;  
float value3 = 123.4f;  
double value4 = 123.4;  
char c = 'a';  
char enter = '\r';  
boolean isNumber = true;  
long value5 = 17L;
```

Q1-2. Which of the options are correct for the following code?

```
public class Prim { // line 1  
    public static void main(String[] args) { // line 2  
        char a = 'a'; // line 3  
        char b = -10; // line 4  
        char c = 'I'; // line 5  
        integer d = 1000; // line 6  
        System.out.println (++a + b++ * c - d); // line 7  
    } // line 8  
} // line 9
```

- a. Code at line 4 fails to compile.
- b. Code at line 5 fails to compile.
- c. Code at line 6 fails to compile.
- d. Code at line 7 fails to compile.

Q1-3. Choose all valid identifiers

- a. int falsetrue;
- b. int javaseminar, javaSeminar;
- c. int DATA-COUNT;
- d. int DATA_COUNT;
- e. int car.count;
- f. int %ctr;
- g. int ¥to£And\$¢;