
Advanced Programming with Java

— Prativa Nyaupane —

Recap of previous class...

1. History of Java
2. Applications of Java
3. Features of Java
4. Architecture of Java

Today's Objectives

- Java Configuration
- Classes
- Objects
- Constructors
- Packages

Java Configuration

Setting up the required configurations

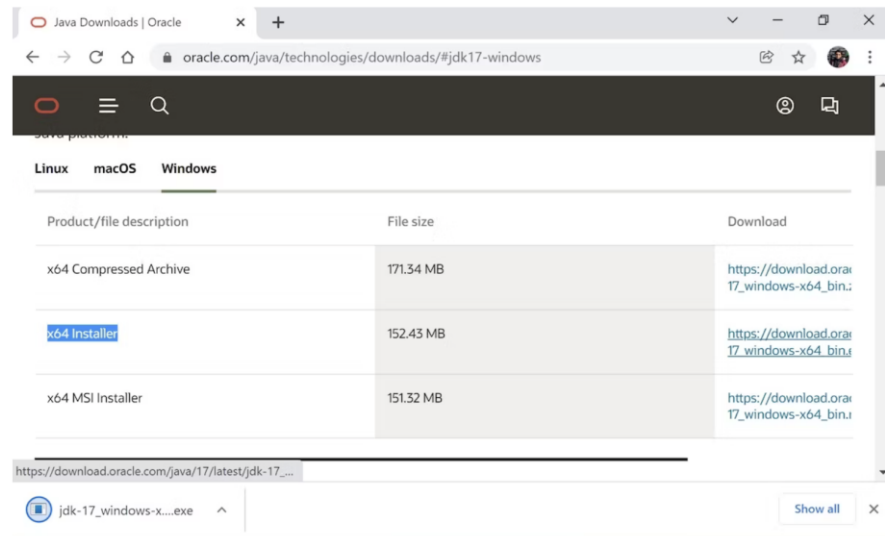
Steps:

1. Install Java in your system - [link](#)
2. Configure Java in your system
3. Download IntelliJ IDEA Community Edition from jetbrains.com - [link](#)
4. Launch IntelliJ and setup initial configurations.

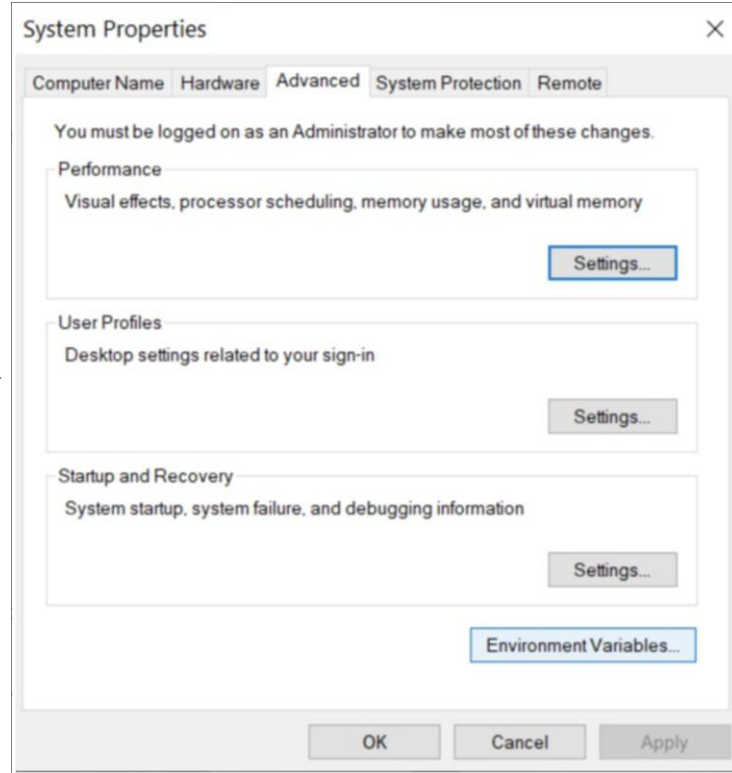
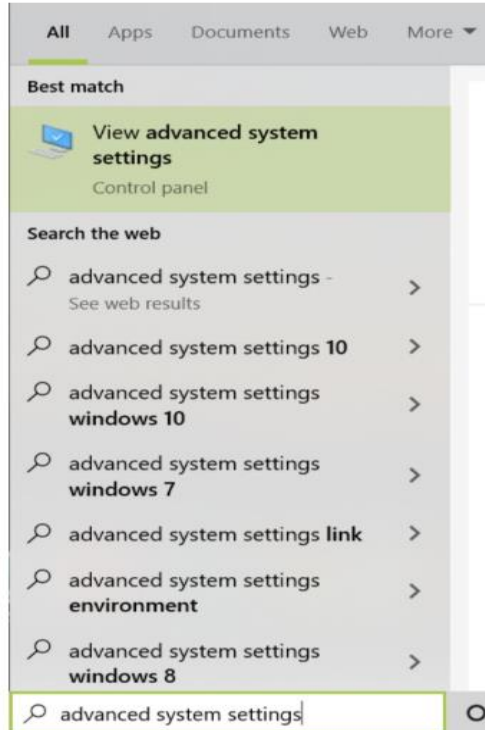
For simplicity, we will be installing IntelliJ in our system. The other popular IDEs for Java programming are NetBeans, Visual Studio Code, XCode, Android Studio....

Set up Java

1. Go to the Java Downloads section of the Oracle website, and download the x64 Installer from there.
2. As soon as the download completes, launch the installation file and click on Yes.
3. The JDK installation wizard will be launched. In the dialog box, click on Next.
4. On the next screen, you'll be asked for the location of your file. Click on Next.
5. Once the JDK installation is complete, click on Close.

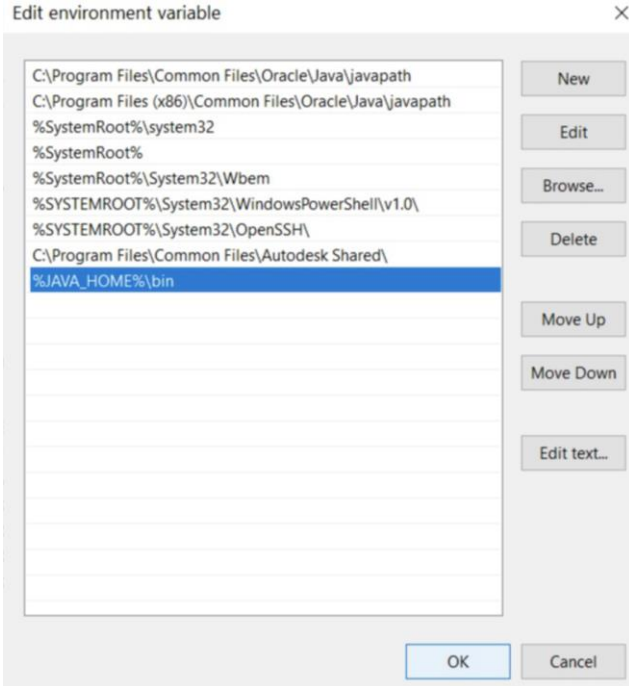


Set up Java path in system



Now, add the following path:

%JAVA_HOME%\bin



Open the terminal and enter the following command:

```
sudo nano /etc/environment.
```

A file will be opened, and add the following command to that file:

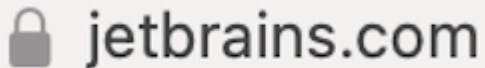
```
JAVA_HOME = "YOUR_PATH".
```

- Replace YOUR_PATH with the JDK bin file path.
- Now restart your computer or virtual machine that you are using (or) reload the file: `source /etc/environment`
- You can test the path by executing

```
echo $JAVA_HOME
```

- If you get the output without any error, then you've set the path correctly.
- If you get any errors, try repeating the procedure again.

Setting up IntelliJ




We're committed to giving back to our wonderful community, which is why IntelliJ IDEA Community Edition is completely free to use

 **IntelliJ IDEA Community Edition**

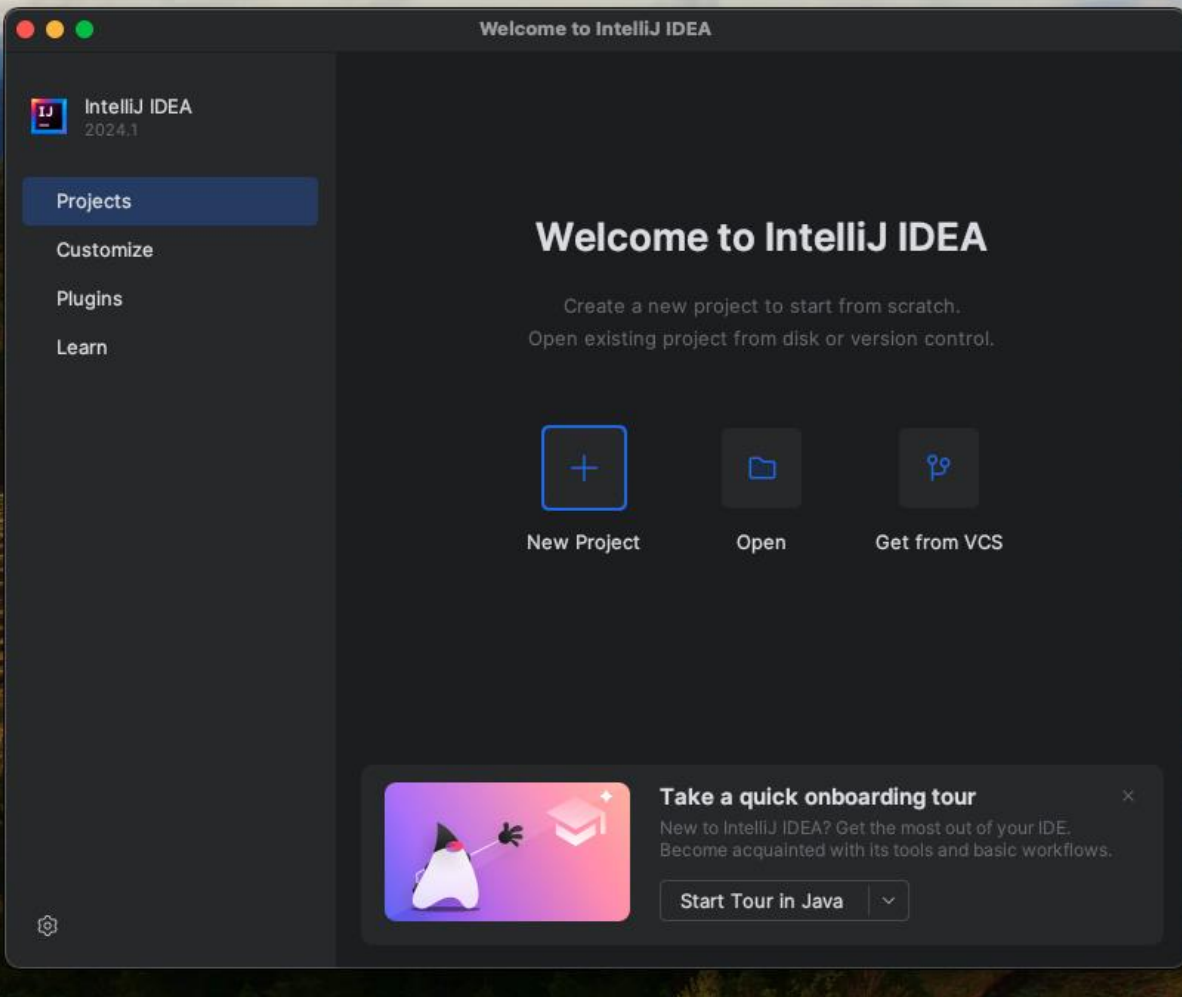
The IDE for Java and Kotlin enthusiasts

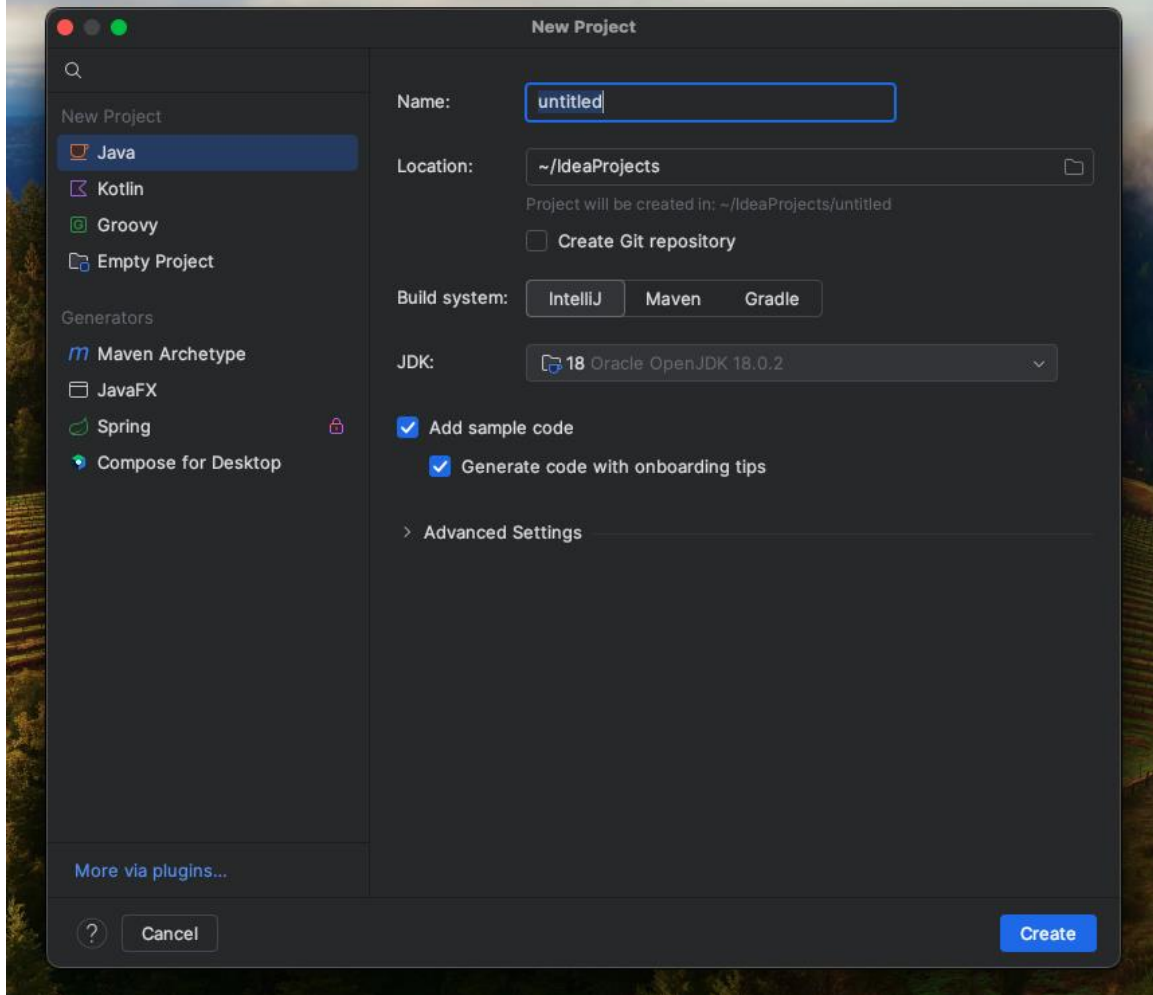
Download .dmg (Intel) ▼

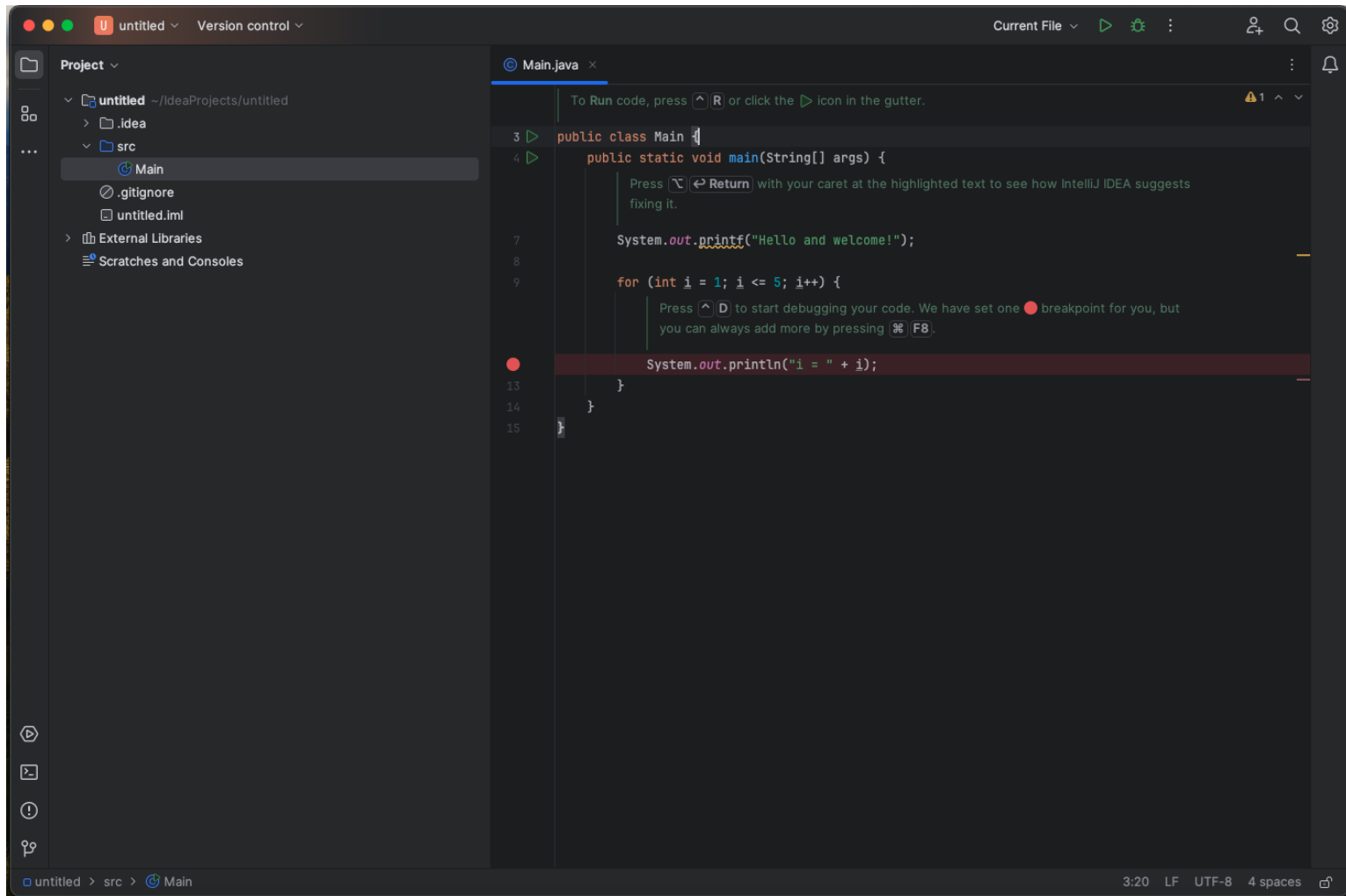
Free, built on open source

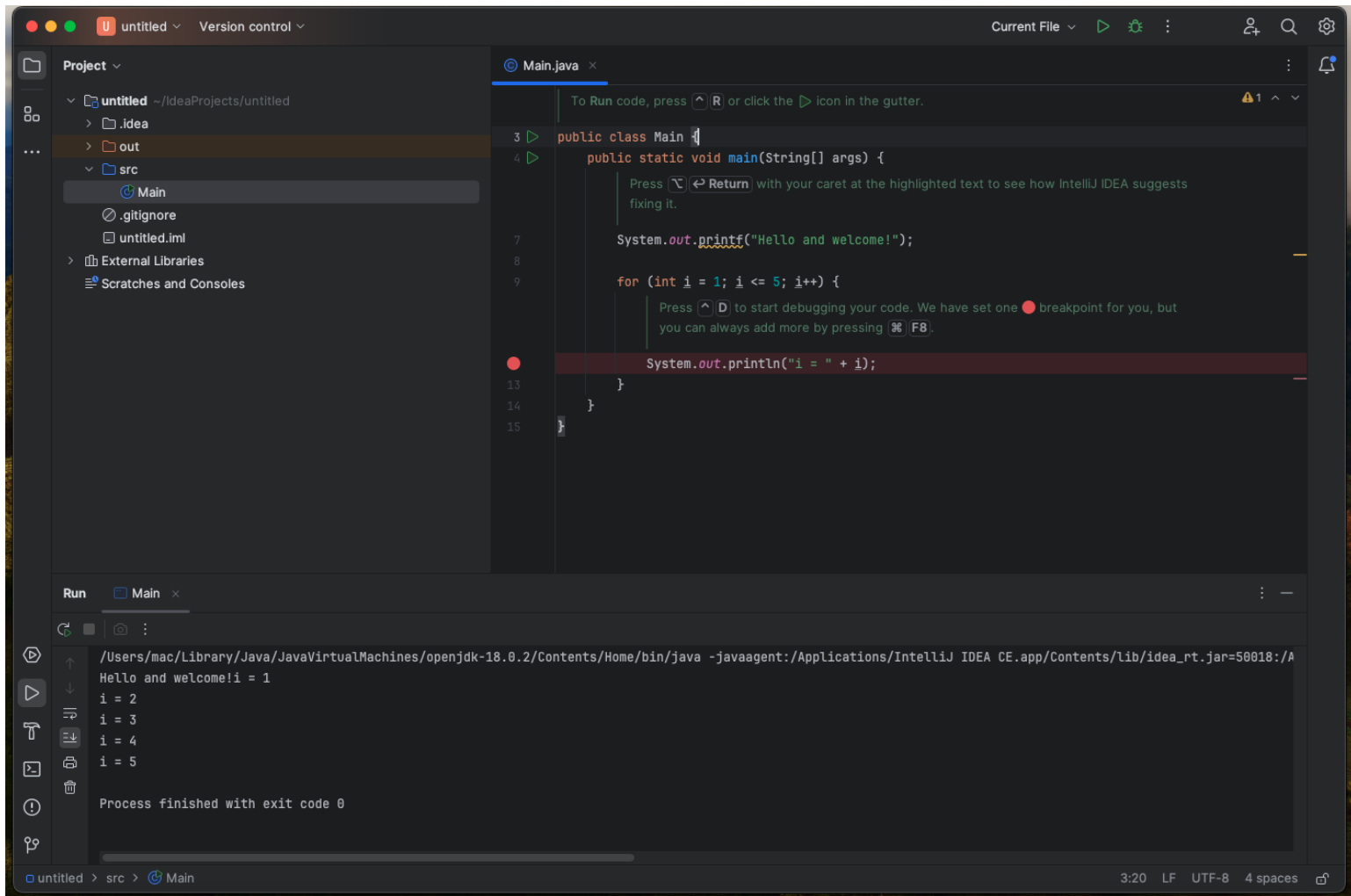
 Select an installer for Intel or Apple Silicon











Class

In Object oriented programming technique, we design a program using objects and classes.

Class is a blueprint for creating objects.

It defines the properties and behaviors(methods or functions) that objects of that class will have.

In this example, Student is the class name. It has four attributes - name, year, semester, rollNumber. It has one operation - displayStudentInformation()

```
public class Student {
    String name;
    int year;
    String semester;
    int rollNumber;

    public Student(int rollNumber, String name, int
year, String semester){
        this.rollNumber = rollNumber;
        this.name = name;
        this.semester = semester;
        this.year = year;
    }

    public void displayStudentInformation(){
        System.out.println("Roll number " +
rollNumber );
        System.out.println("Name " + name );
        System.out.println("Semester " + semester
);
        System.out.println("Year " + year );
    }
}
```

Objects

Once, we have defined class - Student, we can create objects(instances) of that class using the '**new**' operator.

```
Student student = new  
Student(rollNumber,name,year,semester) ;
```

The above code, creates a new 'Student' with their own roll number, name, year and semester.

We can access student's attributes and methods using the dot(' . ') notation.

```
student.displayStudentInformation() ;
```

```
public class Main {  
    public static void main(String[] args) {  
  
        System.out.printf("Start initializing a  
class to create an object");  
  
        int rollNumber = 1;  
        String name = "Ram";  
        int year = 2;  
        String semester = "Fourth";  
  
        Student student = new  
Student(rollNumber,name,year,semester) ;  
        student.displayStudentInformation();  
  
    }  
}
```

Naming conventions of classes and objects

Class Names:

Class names should be nouns, written in UpperCamelCase (also known as PascalCase), where each word in the name starts with an uppercase letter.

Class names should be descriptive and indicative of the purpose or role of the class.

Example: Car, Student, EmployeeDetails.

Object Names:

Object names should be written in lowerCamelCase, where the first letter of the name is lowercase and subsequent words start with uppercase letters.

Object names should be descriptive and indicate the purpose or role of the object.

Example: car, studentInfo, employeeRecord.

Naming convention contd...

Package Names:

Package names should be in lowercase letters.

Package names should follow a reversed domain name structure to ensure uniqueness and avoid naming conflicts.

Example: `com.example.project`.

Why are there naming conventions?

To improve the readability and maintainability of our Java code.

It makes us consistent with the conventions followed by the broader Java community .

Everyone can understand each others code and work in anybody's code.

Class Attributes

Attributes are properties or characteristics that describe the state of an object.

In Java, they are fields within a class.

Objects hold the data associated with each instance of the class.

In Student example, provided earlier, 'rollNumber', 'name', 'semester' and 'year' are attributes of the Student class.

Their values may vary from one object to another.

When we create an instance of a class(an object), we initialize its attributes with specific values.

This allows each object to have its own unique state.

```
public class Main {  
    public static void main(String[] args) {  
  
        System.out.printf("Start initializing a  
class to create an object");  
  
        int rollNumber1 = 1;  
        String name1 = "Ram";  
        int year1 = 2;  
        String semester1 = "Fourth";  
  
        Student student1 = new  
Student(rollNumber1,name1,year1,semester1);  
        student1.displayStudentInformation();  
  
        int rollNumber2 = 1;  
        String name2 = "Ram";  
        int year2 = 2;  
        String semester2 = "Fourth";  
  
        Student student2 = new  
Student(rollNumber2,name2,year2,semester2);  
        student2.displayStudentInformation();  
  
    }  
}
```

```
public class Student {  
    String name;  
    int year;  
    String semester;  
    int rollNumber;  
  
    public Student(int rollNumber, String name, int  
year, String semester){  
        this.rollNumber = rollNumber;  
        this.name = name;  
        this.semester = semester;  
        this.year = year;  
    }  
}
```

Constructors

Constructors in Java are special method used for initializing objects of a class.

They have the same name as the class and are called automatically when an object of the class is created using the 'new' keyword.

Constructors are primarily used to set initial values for the object's attributes.

These can also be used to perform any other necessary setup tasks.

```
public class Student {  
    String name;  
    int year;  
    String semester;  
    int rollNumber;  
  
    public Student(int rollNumber, String name, int  
year, String semester){  
        this.rollNumber = rollNumber;  
        this.name = name;  
        this.semester = semester;  
        this.year = year;  
    }  
}
```


Types of Constructors

Default Constructor

It has no parameters.

If you do not define any constructor in your class, Java provides a default constructor automatically.

```
public class Student {  
    String name;  
    int year;  
    String semester;  
    int rollNumber;  
  
    public Student() {  
        System.out.println("This is a default  
constructor");  
    }  
}
```

This constructor is called using:

```
Student student1 = new Student();
```

Parameterized Constructor

It takes parameters to initialize the object's attributes with specific values.

```
public Student(int rollNumber, String name, int  
year, String semester) {  
    this.rollNumber = rollNumber;  
    this.name = name;  
    this.semester = semester;  
    this.year = year;  
}
```

This constructor is called as follows:

```
Student student2 = new  
Student(rollNumber, name1, year1, semester1);  
  
student1.displayStudentInformation();
```

Packages

A package is a way to organize related classes, interfaces and sub-packages into a hierarchical structure.

It helps in organizing and managing large codebases, providing a way to create a modular and hierarchical structure for classes and preventing naming conflicts.

It provides a means of grouping and managing classes and enhances code modularity and reusability.

