MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Dr. Vishwanath Karad

MIT WORLD PEACE UNIVERSITY | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

CET1042B: Object Oriented Programming with C++ and Java

SCHOOL OF COMPUTER ENGINEERING AND TECHNOLOGY

S. Y. B. TECH. COMPUTER SCIENCE AND ENGINEERING

(CYBERSECURITY AND FORENSICS)

CET1042B: Object Oriented Programming with C++ and Java

Teaching Scheme	**Credits: 02 + 02 = 04**
Theory: 2 Hrs. / Week	**Practical:** 4 Hrs./Week

Course Objectives

1) **Knowledge:** (i) Learn object oriented paradigm and its fundamentals.
2) **Skills:** (i) Understand Inheritance, Polymorphism and dynamic binding using OOP.
   (ii) Study the concepts of Exception Handling and file handling using C++ and Java.
3) **Attitude:** (i) Learn to apply advanced concepts to solve real world problems.

Course Outcomes

1) Apply the basic concepts of Object Oriented Programming to design an application.
2) Make use of Inheritance and Polymorphism to develop real world applications.
3) Apply the concepts of exceptions and file handling to store and retrieve the data.
4) Develop efficient application solutions using advanced concepts.

Assignment 1

Demonstrate the use of objects, classes, constructors and destructors using C++ and JAVA.

Practice Program

An airline information system want to maintain the information of passengers travelling by their airways. Following is the information that is to be maintained for the passengers.

- Name of passenger
- Age of passenger
- flight no.
- departure time
- source
- destination

Design a C++ class to accept and display information for the airlines. Demonstrate the use of array of objects concept.

Problem Statement_Assignment 1_Using C++

Develop an object-oriented program to create a database of Employee Information Management System containing the following information:

- Employee Name
- Employee number
- Qualification
- Address
- Contact Number
- Salary details (basic, DA, TA, Net salary), etc.

Create a class Named “**Employee**”, construct the database with suitable member functions for initializing and destroying the data viz. Default constructor, Parameterized constructor and Copy constructor, and destructor. Use dynamic memory allocation concept while creating and destroying the object of a class. Use static data member concept wherever required. Accept and Display the information of Employees.

Getting Introduced with C++ Programming Paradigms

Simple C++ Program

```
// Simple C++ program to display "Hello World"
// Header file for input output functions
#include<iostream>
```

instructs the compiler to include the contents of the file enclosed within angular brackets into the source file.

```
using namespace std;
```

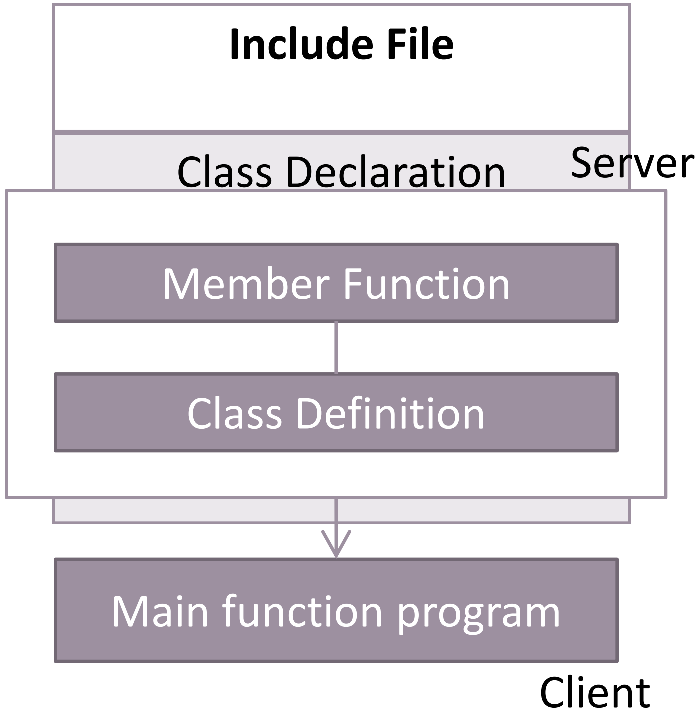
defines a scope for the identifiers that are used in a program

```
// main function - where the execution of program begins
int main()
{
   // prints message as hello world
   cout<<"Hello World";

   return 0;
}
```

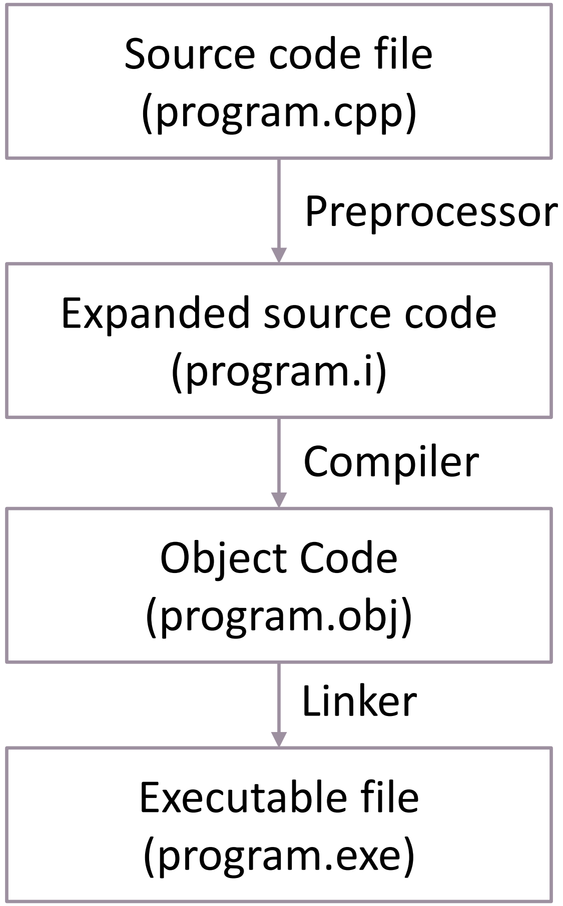
every main() returns an integer value to operating system and therefore it should end with return (0) statement

Structure of C++ Program

- Typical C++ program contains four sections
- It is a common practice to organize a program into three separate files
- The class declarations are placed in a header file and the definitions of member functions go into another file.
- The main program that uses the class is placed in a third file which “**includes**” the previous two files as well as any other file required



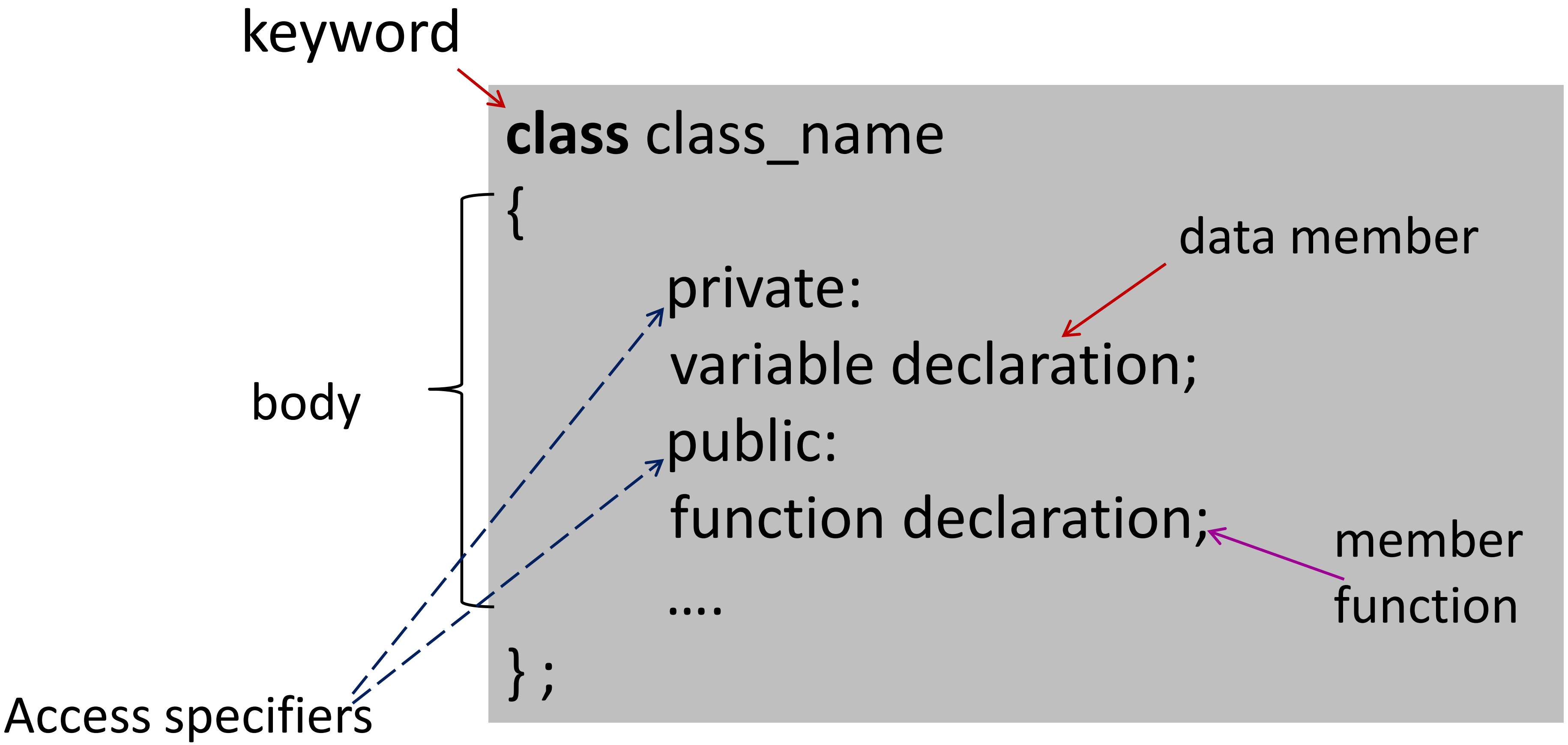
C++ Compilation and Linking



Class

- A way to bind data and associated function together
- An expanded concept of a data structure, instead of holding only data, it can hold both data and function.
- The data is to be hidden from external use.

Syntax of Class



keyword

```
class class_name
{
            private:
            variable declaration;
            public:
            function declaration;
            ….
} ;
```

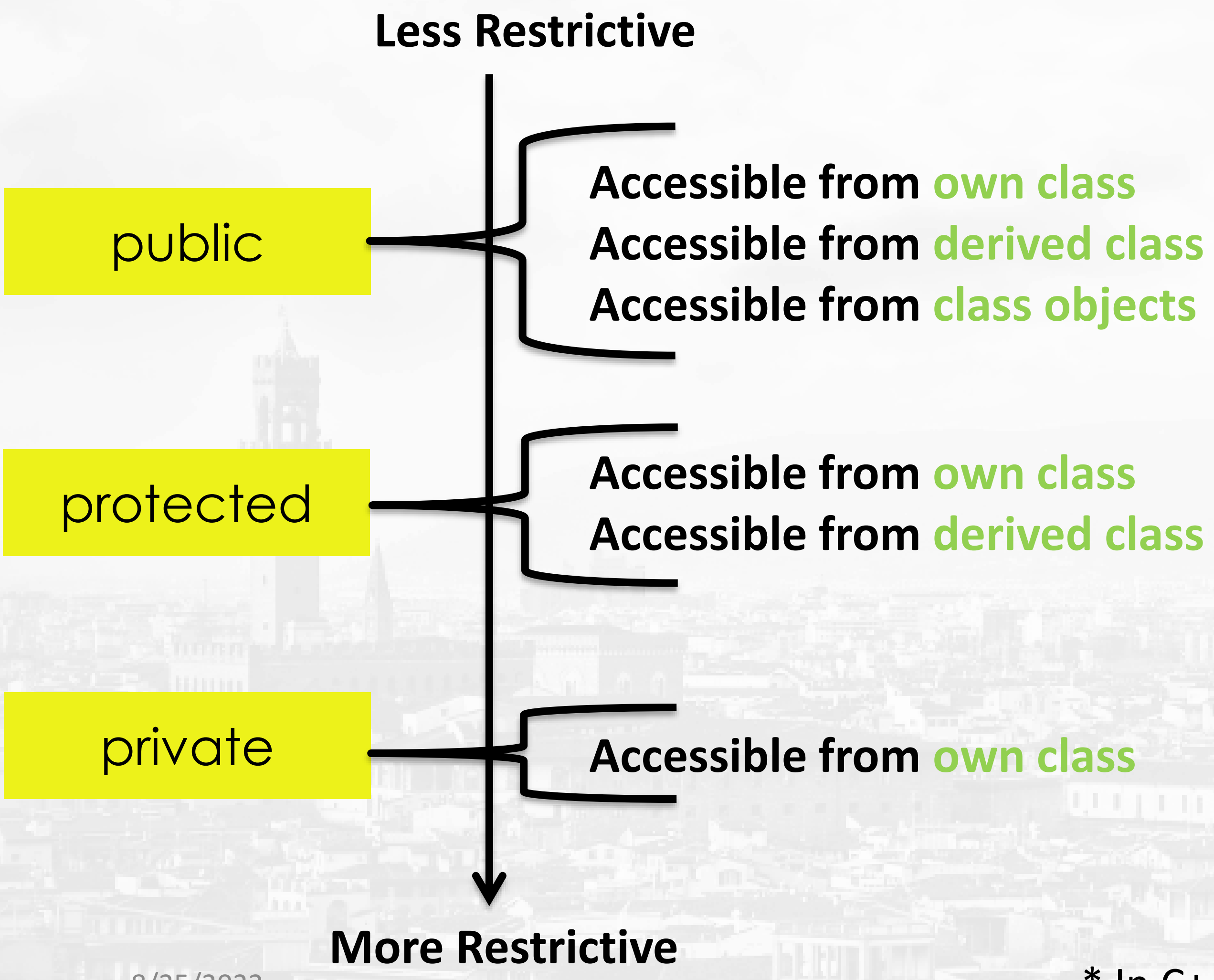
data member

member function

body

Access specifiers

Access Specifiers



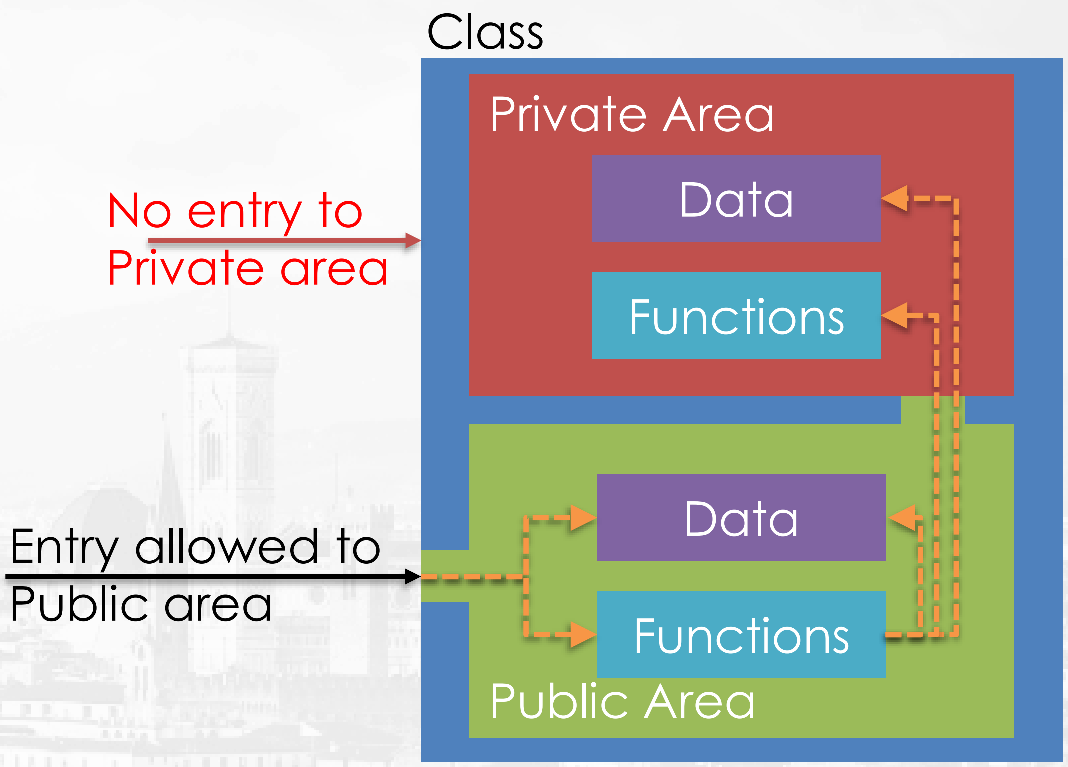
Less Restrictive

public
- Accessible from own class
- Accessible from derived class
- Accessible from class objects

protected
- Accessible from own class
- Accessible from derived class

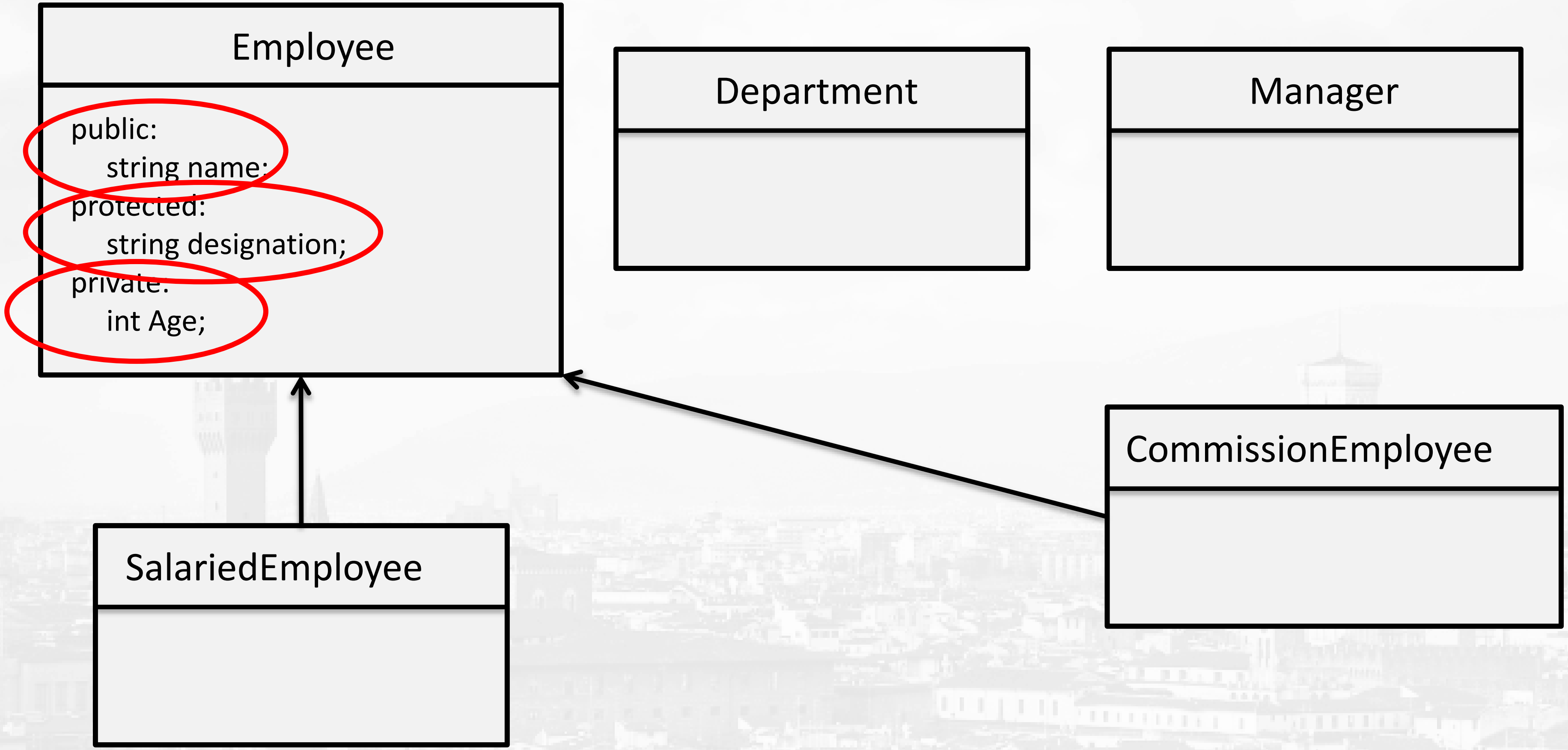
private
- Accessible from own class

More Restrictive



* In C++ By default all items defined in the class are private

Access Specifiers



Employee

public:
    string name;
protected:
    string designation;
private:
    int Age;

Department

Manager

CommissionEmployee

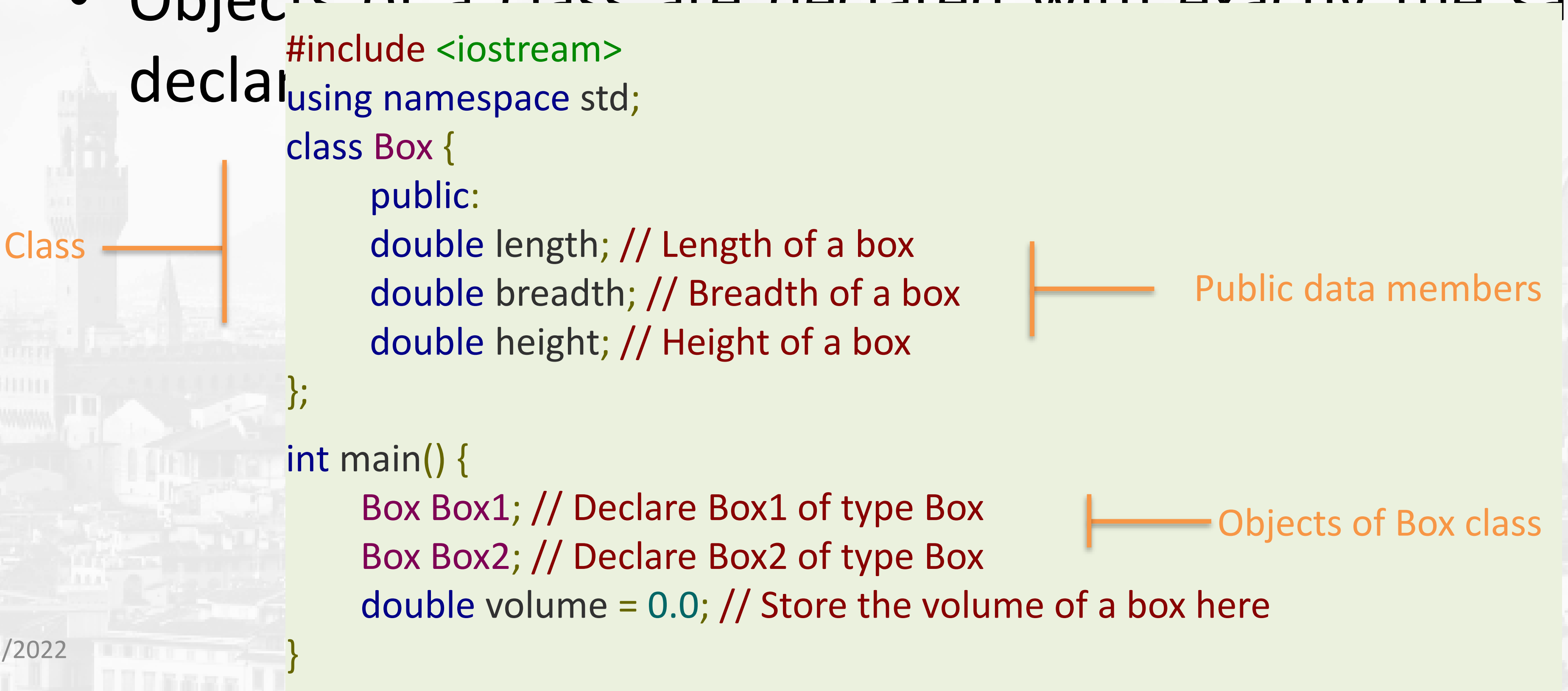
SalariedEmployee

Access Specifiers

```
class Person {
    public:                              //access control
        string firstName;     //these data members
        string lastName;      //can be accessed
        int dateofBirth;      //from anywhere
    private:
        string address;   // can be accessed inside the class
        long int insuranceNumber;   //and by friend classes/functions
    protected:
        string phoneNumber; // can be accessed inside this class,
        int salary;     // by friend functions/classes and derived classes

};
```

Objects

- A class provides the blueprints for objects, so basically an object is created from a class.
- Objects of a class are declared with exactly the same sort of declar



```
#include <iostream>
using namespace std;
class Box {
        public:
        double length; // Length of a box
        double breadth; // Breadth of a box
        double height; // Height of a box
};

int main() {
       Box Box1; // Declare Box1 of type Box
       Box Box2; // Declare Box2 of type Box
       double volume = 0.0; // Store the volume of a box here
}
```

Class

Public data members

Objects of Box class

Objects

- The objects of class will have their own copy of data members.
- The public data members of objects of a class can be accessed using the direct member access operator (.)

```
int main() {
      Box Box1; // Declare Box1 of type Box
      Box Box2; // Declare Box2 of type Box
      double volume = 0.0; // Store the volume of a box here

      // box 1 specification
     Box1.height = 5.0;
     Box1.length = 6.0;
     Box1.breadth = 7.0;

     // box 2 specification
     Box2.height = 10.0;
     Box2.length = 12.0;
     Box2.breadth = 13.0;

      // volume of box 1
      volume = Box1.height * Box1.length * Box1.breadth;
      cout << "Volume of Box1 : " << volume <<endl;

      // volume of box 2
      volume = Box2.height * Box2.length * Box2.breadth;
      cout << "Volume of Box2 : " << volume <<endl;
      return 0;
}
```

Access data members of **Box1** object

Access data members of **Box2** object

Member Functions

- Can be defined inside class

```
return_type  function_name  (parameters)
{
        // function body
}
```

- Or outside the class

```
return_type  class_name::function_name (forma
{
        // function body
}
```

- Functions defined inside class are treated as inline functions by compiler

```
class Box {
  public:
    double length, breadth, height;
    double getVolume(void) {   //  Returns box volume
        return length * breadth * height;
    }
    double getSurfaceArea(void);    // returns surface area
};
// member function definition
double Box::getSurfaceArea(void) {
     ….
}
int main() {
     Box Box1;
     Box1.length = 10;
     Box1.height = 20;
     Box1.breadth=30;
     cout << “Volume of box: “ << Box1.getVolume() << endl;
     cout << “Surface Area of box: “ << Box1.getSurfaceArea() <<
endl;
}
```

Inline function

member function declaration

member function definition outside class

accessing member functions

Arrays of Objects

- **Several** objects of the **same class** can be declared as an array and used just like an array of any other data type.

- The **syntax** for declaring and using an object array is **exactly** the **same** as it is for any other type of array.

Array of Objects

```
#include <iostream>
#include <string>
using namespace std;
class Student
{               string name;
                int marks;
                public:
                void  getdata()
                {
                                cout<<"enter name";
                                cin>> name;
                                cout <<"enter marks";
                                cin>>marks;
                }
                void putdata()
                {               cout << "Name : " << name << endl;
                                cout << "Marks : " << marks << endl;
                }
};
```

```
int main()
{               Student st[5];
                for( int i=0; i<5; i++ )
                {
                cout << "Student " << i + 1 << endl;
                st[i].getdata();
                }
                for( int i=0; i<5; i++ )
                {
                cout << "Student " << i + 1 << endl;
                st[i].putdata();
                }
                return 0;
}
```

Practice Program Snippet using C++

```
#include<iostream>
#include<string.h>
using namespace std;
class airline
{
public:
     string pname;
     int age, flightno;
     int deptime;
     string source;
     string destination;

void getdata()
{
   cout<<"enter passenger name";
  getline(cin, pname);
  cin.ignore();
  cout<<"enter age";
  cin>>age;
```

```
cout<<"enter flight no";
Cin>>flightno;
cout<<"enter departure time";
cin>>deptime;
cout<<"enter source location";
 getline(cin, source);
 cin.ignore();
cout<<"enter destination location";
 getline(cin, destination);
 cin.ignore();
}
```

Practice Program Snippet using C++ Continued

```
void putdata()
{
cout<<“passenger name is "<<pname<<endl;
cout<<“age of passenger"<<age<<endl;
cout<<“Flight no"<<flightno<<endl;
cout<<“Departure time  "<<deptime<endl;
cout<<“source location"<<source<<endl;
cout<<“destination location"<<destination<<endl;
}
};

int main()
{
 int i;
airline p[10];
cout<<"Enter passenger details"<<endl;
```

```
int n;
cout<<"Enter number of passengers";
cin>>n;
for(i=0;i<n;i++)
{
cout<<“Enter passengers data:";
p[i].getdata();
}
for(i=0;i<n;i++)
{
cout<<“passenger details:";
p[i].putdata();
}
return 0;
}
```

Assignment 1_Problem Statement

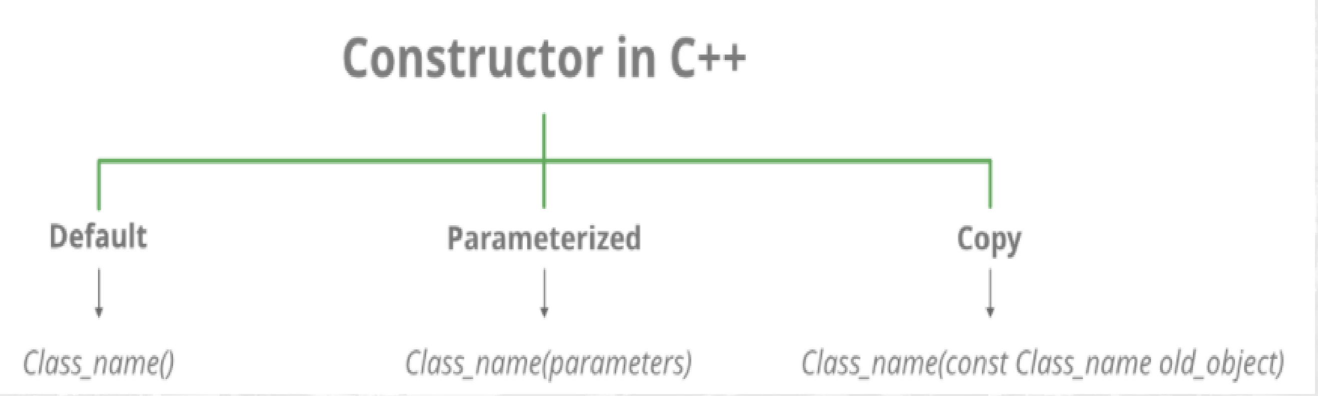
Develop an object-oriented program to create a database of Employee Information Management System containing the following information:

- Employee Name
- Employee number
- Qualification
- Address
- Contact Number
- Salary details (basic, DA, TA, Net salary), etc.

Create a class Named “**Employee”**, construct the database with suitable member functions for initializing and destroying the data viz. constructor, default constructor, Copy constructor, and destructor. Use dynamic memory allocation concept while creating and destroying the object of a class. Use static data member concept wherever required. Accept and Display the information of Employees.

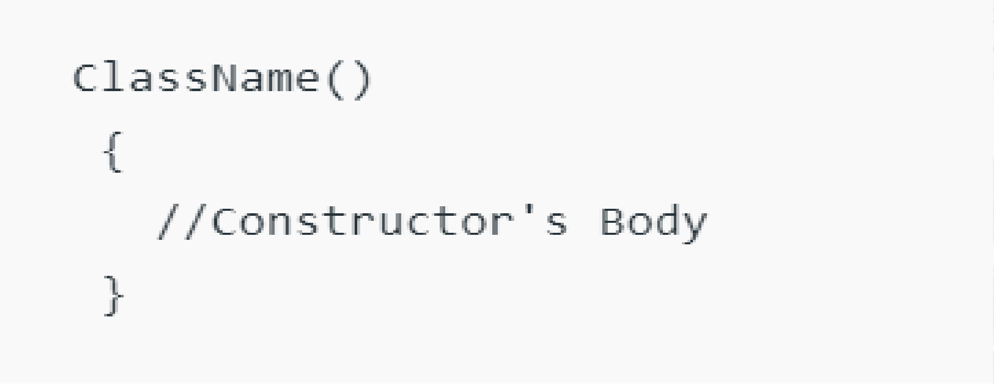
Constructor

- A constructor is a special type of member function of a class which initializes objects of a class.
- Constructor is automatically called when object(instance of class) is created.
- It is special member function of the class because it does not have any return type.
- **How constructors are different from a normal member function?**
  - Constructor has same name as the class itself
  - Constructors don’t have return type
  - A constructor is automatically called when an object is created.
  - It must be placed in public section of class.
  - If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body).



Constructor

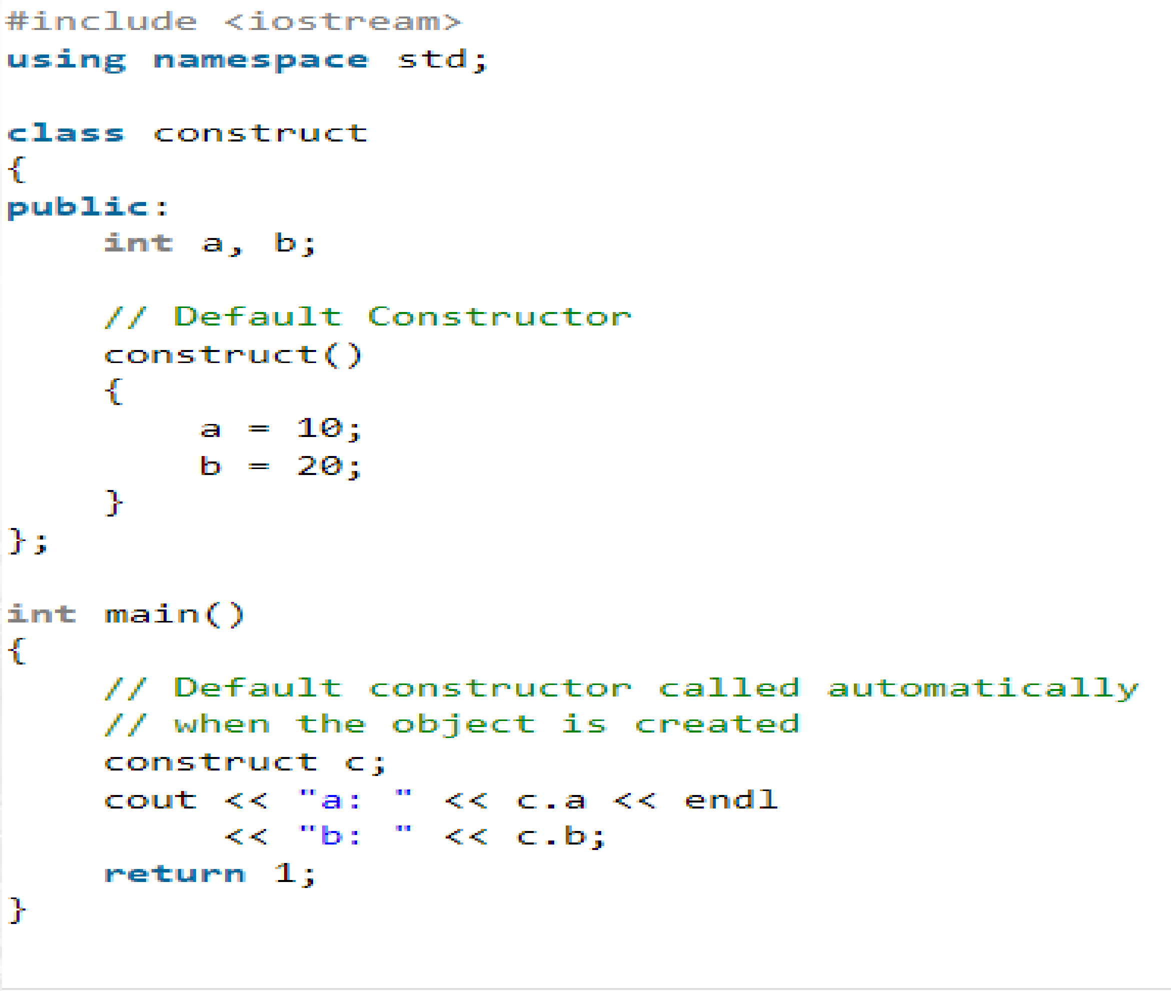
- A constructor is a member function of a class that has the same name as the class name.
- It helps to initialize the object of a class.
- It can either accept the arguments or not.
- It is used to allocate the memory to an object of the class. It is called whenever an instance of the class is created. It can be defined manually with arguments or without arguments.
- There can be many constructors in class. It can be overloaded but it can not be inherited or virtual.
- There is a concept of copy constructor which is used to initialize a object from another object.
- **Syntax**



```
ClassName()
 {
   //Constructor's Body
 }
```

Default Constructor

- Default constructor is the constructor which doesn’t take any argument. It has no parameters.
- Example

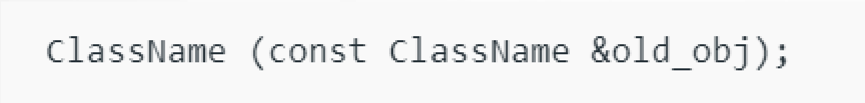


```
#include <iostream>
using namespace std;

class construct
{
public:
    int a, b;

    // Default Constructor
    construct()
    {
        a = 10;
        b = 20;
    }
};

int main()
{
    // Default constructor called automatically
    // when the object is created
    construct c;
    cout << "a: " << c.a << endl
         << "b: " << c.b;
    return 1;
}
```

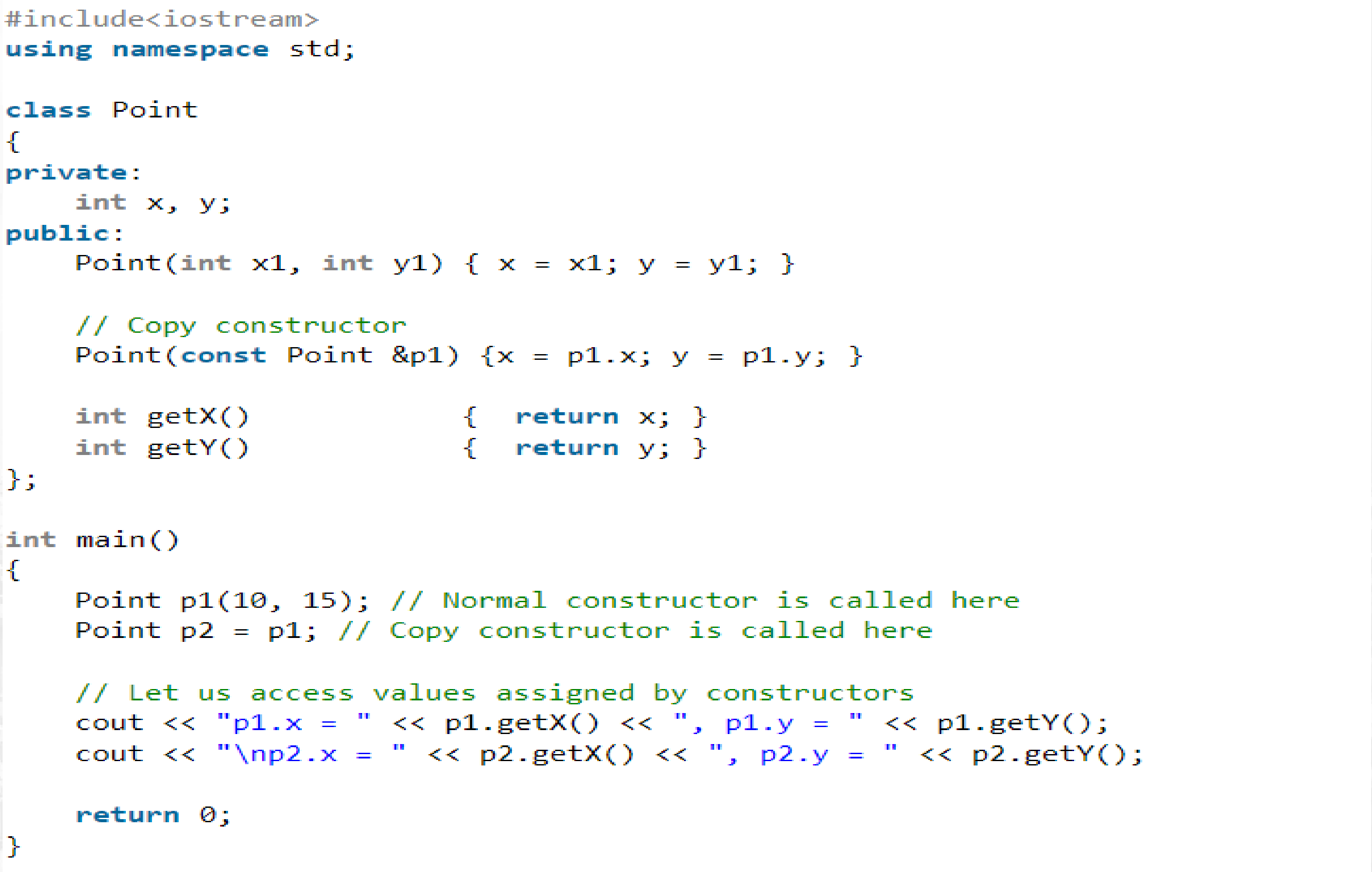
Copy Constructor

- There is a concept of copy constructor which is used to initialize a object from another object.

Syntax



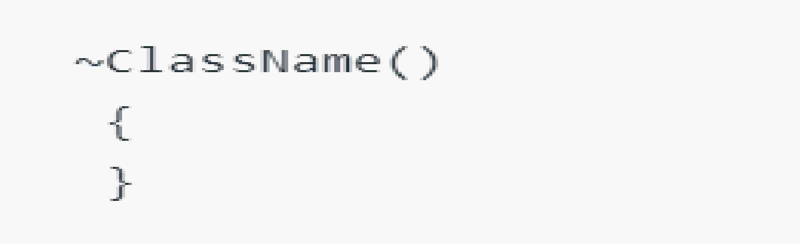
```
ClassName (const ClassName &old_obj);
```



```
#include<iostream>
using namespace std;

class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1) { x = x1; y = y1; }

    // Copy constructor
    Point(const Point &p1) {x = p1.x; y = p1.y; }

    int getX()              {  return x; }
    int getY()              {  return y; }
};

int main()
{
    Point p1(10, 15); // Normal constructor is called here
    Point p2 = p1; // Copy constructor is called here

    // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();

    return 0;
}
```

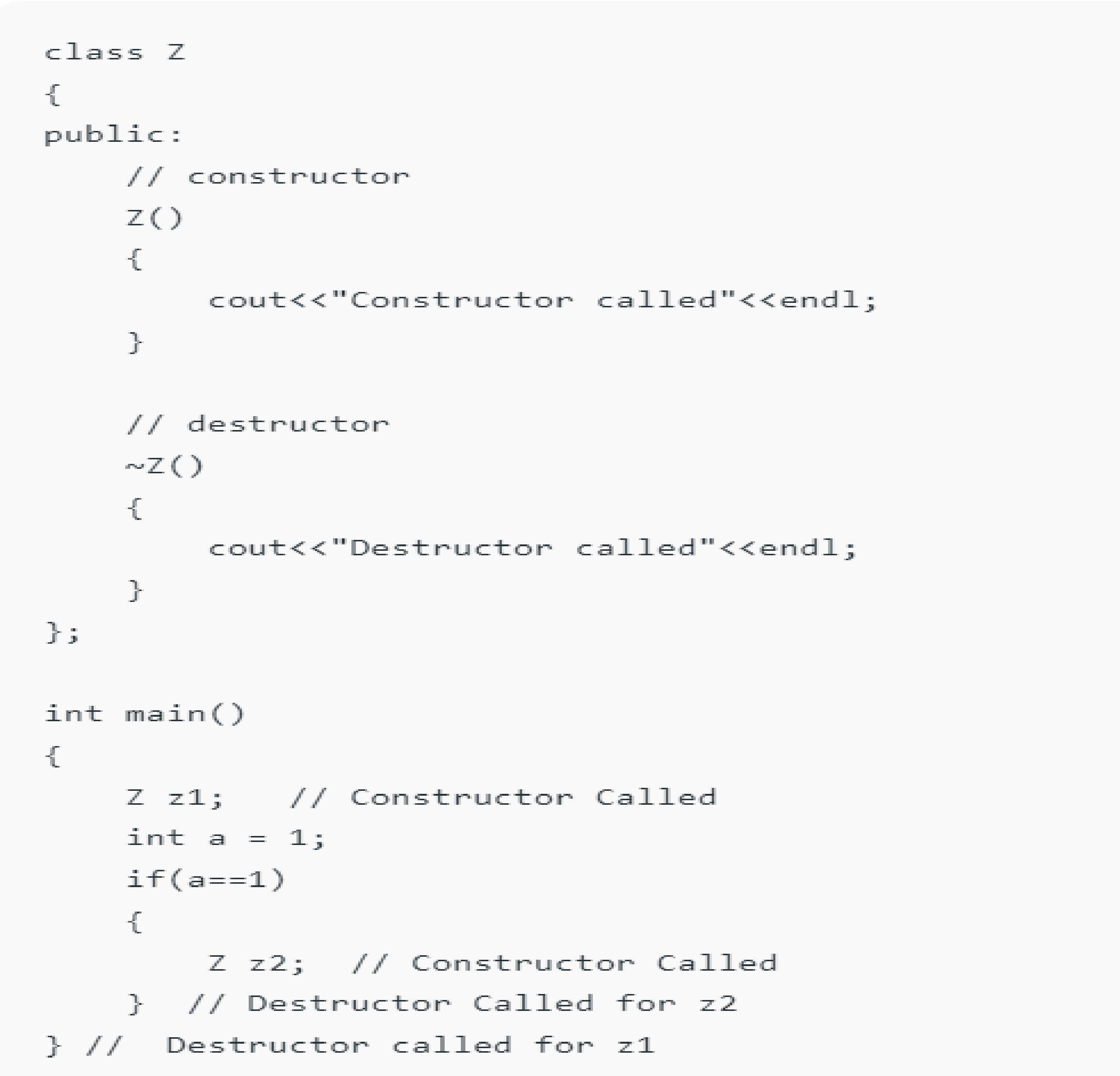
Destructor

- Like constructor, de constructor is also a member function of a class that has the same name as the class name preceded by a tilde(~) operator.
- It helps to de allocate the memory of an object. It is called while object of the class is freed or deleted.
- In a class, there is always a single destructor without any parameters so it can’t be overloaded.
- It is always called in the reverse order of the constructor. if a class is inherited by another class and both the classes have a destructor then the destructor of the child class is called first, followed by the destructor of the parent or base class.



```
~ClassName()
 {
 }
```

Example of Destructor



```
class Z
{
public:
    // constructor
    Z()
    {
        cout<<"Constructor called"<<endl;
    }

    // destructor
    ~Z()
    {
        cout<<"Destructor called"<<endl;
    }
};

int main()
{
    Z z1;   // Constructor Called
    int a = 1;
    if(a==1)
    {
        Z z2;  // Constructor Called
    }  // Destructor Called for z2
} //  Destructor called for z1
```

Static Member

- We can define class members static using **static** keyword.
- When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.
- A static member is shared by all objects of the class.
- All static data is initialized to zero when the first object is created, if no other initialization is present.
- We can't put it in the class definition but it can be initialized outside the class by re declaring the static variable, using the scope resolution operator **::** to identify which class it belongs to.
- By declaring a function member as static, you make it independent of any particular object of the class.
- A static member function can be called even if no objects of the class exist and the **static** functions are accessed using only the class name and the scope resolution operator **::**

Assignment 1_Algorithm

1. Start
2. Create Employee class
3. Declare appropriate data members and define member functions
4. Accept multiple employee data using array of object
5. Show usage of default constructor and display data
6. Show usage of parameterized constructor and display data
7. Show usage of copy constructor and display data
8. End

Getting Introduced with Java Programming Paradigms

Practice Assignment 1

Demonstration of Java program to print the text

“Welcome to Java Programming World”

Problem Statement_Assignment 1_Using Java

Develop an object-oriented program to create a database of Employee Information Management System containing the following information:

- Employee Name
- Employee number
- Qualification
- Address
- Contact Number
- Salary details (basic, DA, TA, Net salary), etc.

Create a class Named “**Employee**”, construct the database with suitable member functions for initializing the data viz. default constructor, parameterized constructor and Copy constructor etc. Accept and Display the information of Employees.

Write Once,
Run
Anywhere

Java Programming

- Java is a high level, object-oriented, class-based, concurrent, secured and general-purpose computer-programming language.
- Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995.
- There are 4 platforms or editions of Java:
  - Java SE (Java Standard Edition)
  - Java EE (Java Enterprise Edition)
  - Java ME (Java Micro Edition)
  - JavaFX

It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform.

It is a micro platform that is dedicated to mobile applications.

It is used to develop rich internet applications. It uses a lightweight user interface API

Java vs C++ Programming

Comparison Parameter	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.

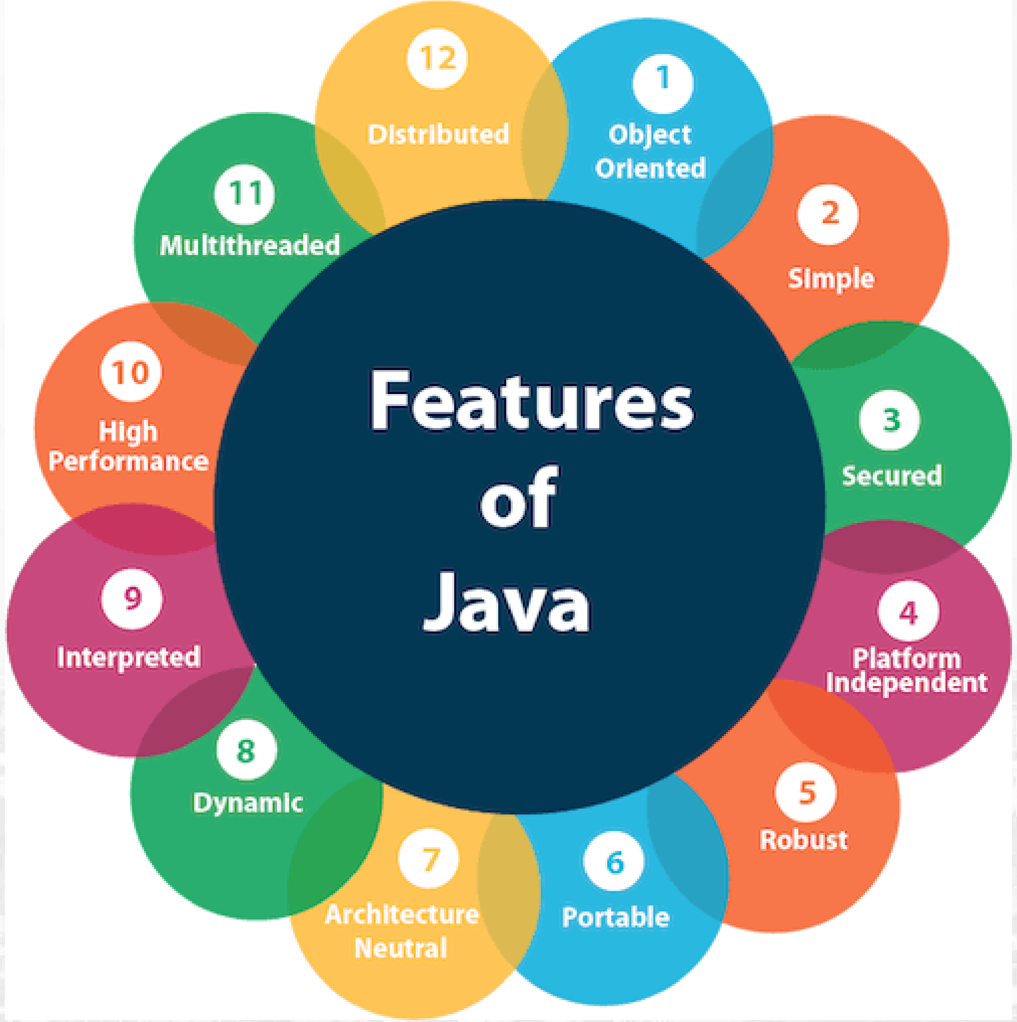
Java vs C++ Programming

Comparison Parameter	C++	Java
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.

Java vs C++ Programming

Comparison Parameter	C++	Java
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Features of Java



Software Development Kit (SDK)

- Collection of software development tools in one installable package.
- Facilitates the creation of applications by having a compiler, debugger and perhaps a software framework.
- Specific to a hardware platform and operating system combination.
- To create applications with advanced functionalities such as advertisements, push notifications, etc; most application software developers use specific software development kits.
- For example – Android NDK, iOS SDK, Java Development Kit (JDK), Java Web Services Development Pack, Microsoft Windows SDK, Windows App SDK

Java Development Kit (JDK)

- The JDK is a development environment for building applications and components using the Java programming language.

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

- Latest Version – jdk 17
  - available at https://docs.oracle.com/en/java/javase/index.html

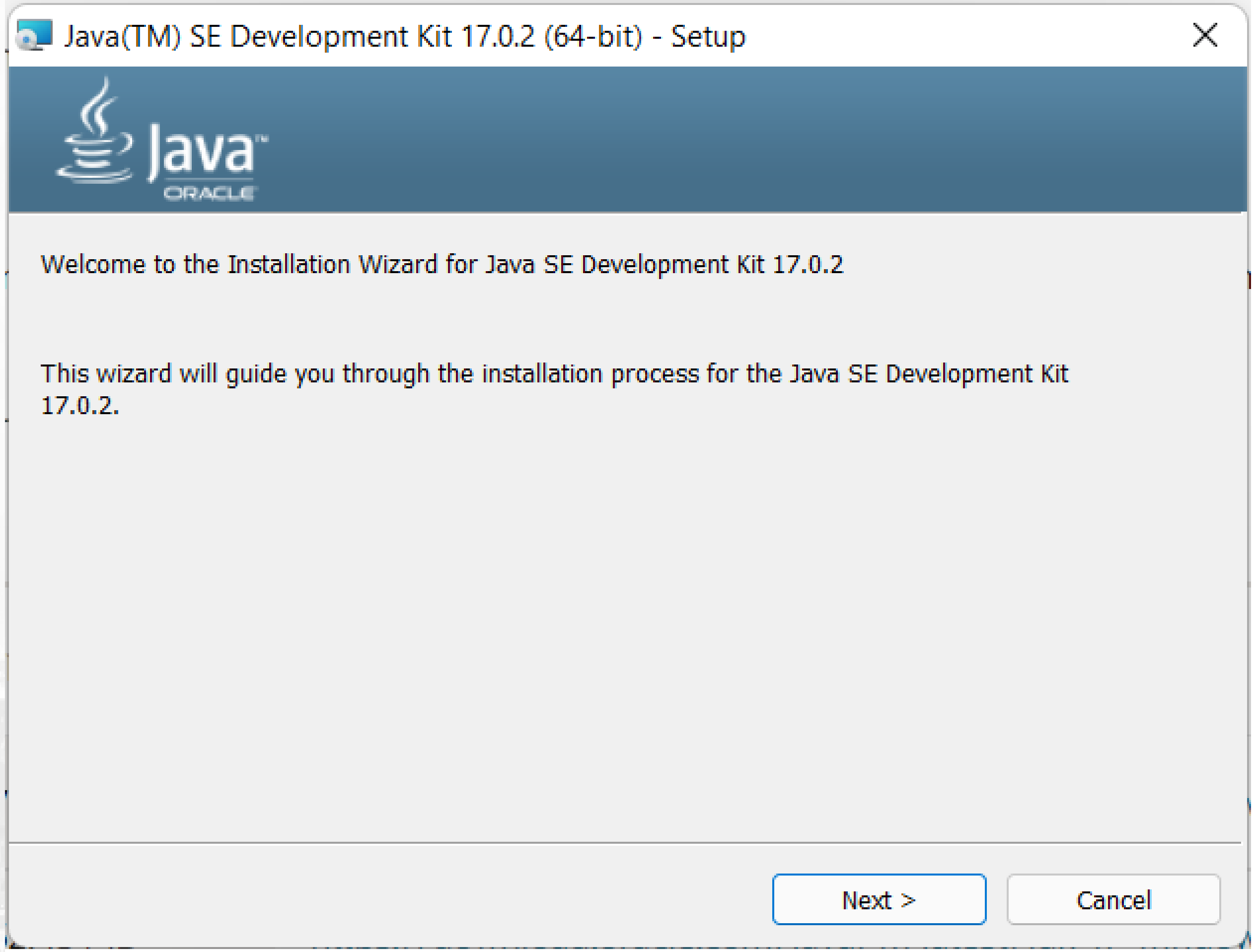
Java Virtual Machine (JVM)

- JVM is an abstract machine.
- It is a specification that provides runtime environment in which java bytecode can be executed.
- JVM is platform dependent.
- A specification where working of Java Virtual Machine is specified.
  - But, implementation provider is independent to choose the algorithm.
  - Its implementation has been provided by Oracle and other companies.
- An implementation is known as JRE (Java Runtime Environment).
- Runtime Instance – Whenever java command is written on the command prompt to run the java class, an instance of JVM is created.

Installation Steps for JDK on Windows

- **Download the JDK available at https://docs.oracle.com/en/java/javase/index.html**
- **Install it**
- Create the Java program
- Compile and run the Java program

Installation Steps for JDK

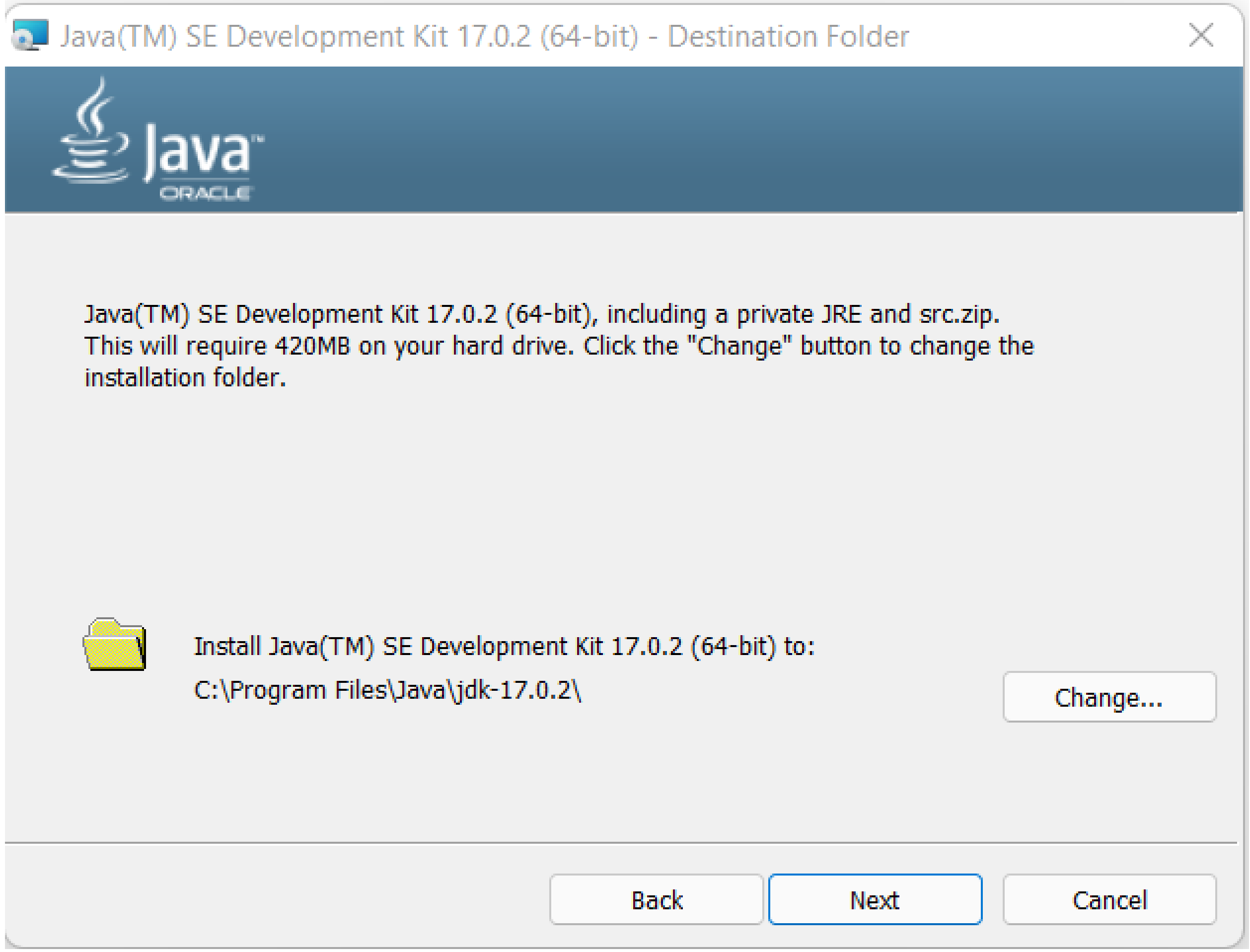


Java(TM) SE Development Kit 17.0.2 (64-bit) - Setup

Welcome to the Installation Wizard for Java SE Development Kit 17.0.2

This wizard will guide you through the installation process for the Java SE Development Kit 17.0.2.

Installation Steps for JDK

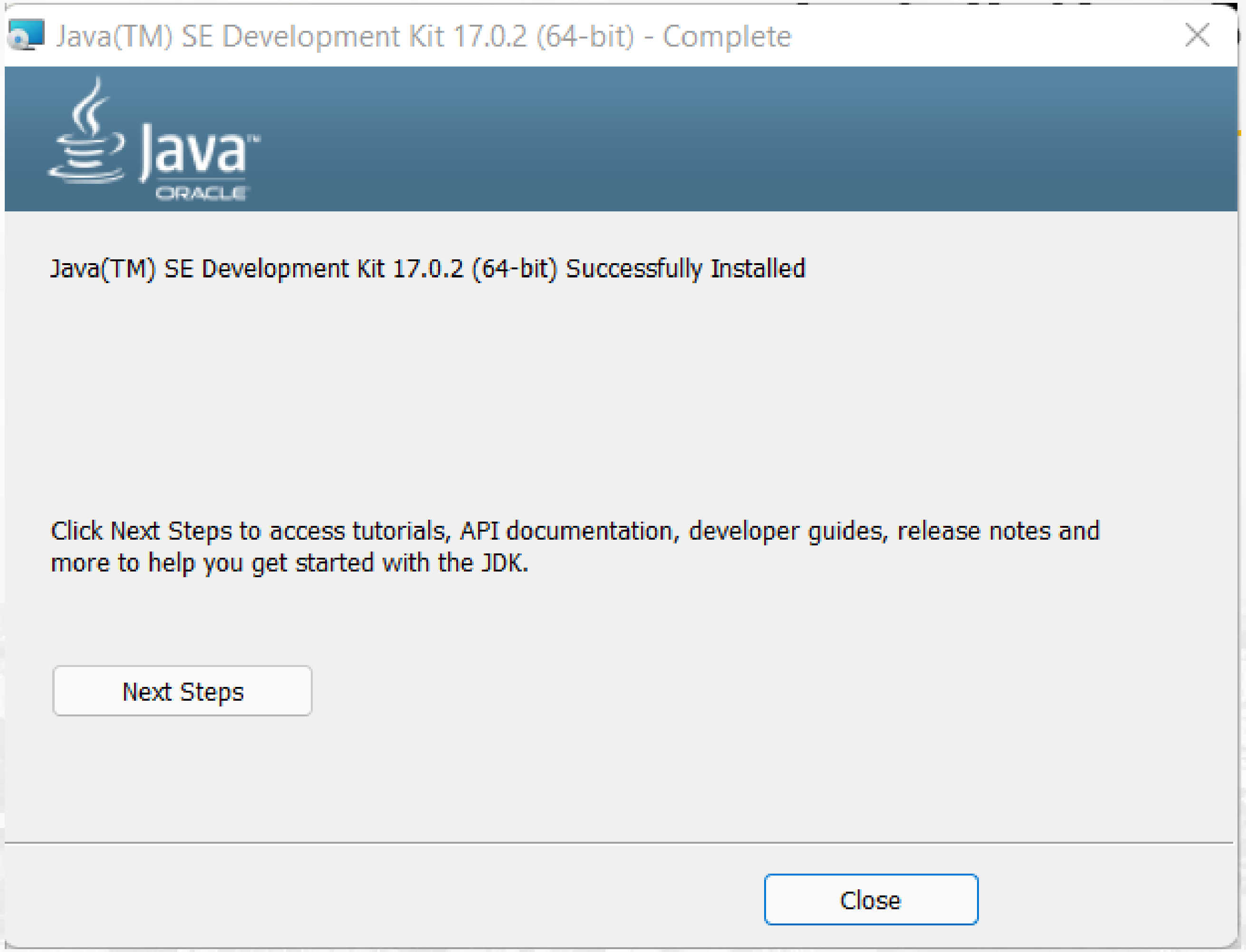


Java(TM) SE Development Kit 17.0.2 (64-bit) - Destination Folder

Java(TM) SE Development Kit 17.0.2 (64-bit), including a private JRE and src.zip. This will require 420MB on your hard drive. Click the "Change" button to change the installation folder.

Install Java(TM) SE Development Kit 17.0.2 (64-bit) to:
C:\Program Files\Java\jdk-17.0.2\

Installation Steps for JDK



Java(TM) SE Development Kit 17.0.2 (64-bit) - Complete

Java(TM) SE Development Kit 17.0.2 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

Installation Steps for JDK on Windows

- Download the JDK available at https://docs.oracle.com/en/java/javase/index.html
- Install it
- **Create the Java program**
- Compile and run the Java program

Create the Java program

Open the notepad

```
public class MyFirstJavaProgram {


   /* This is my first java program.
    * This will print ' Welcome to Java Programming World ' as the output
    */
   public static void main(String []args) {
      System.out.println("Welcome to Java Programming World");
   }
}
```

For all class names the first letter should be in Upper Case.

Stores Java command-line arguments and is an array of type java.lang.String class.

Java program processing starts from the main() method

Java **main()** method is always static, so that compiler can call it without the creation of an object or before the creation of an object of the class.

Save the file as “MyFirstJavaProgram.java”

Every Java program has a class name that must match the filename and every program must contain the main() method

Installation Steps for JDK on Windows

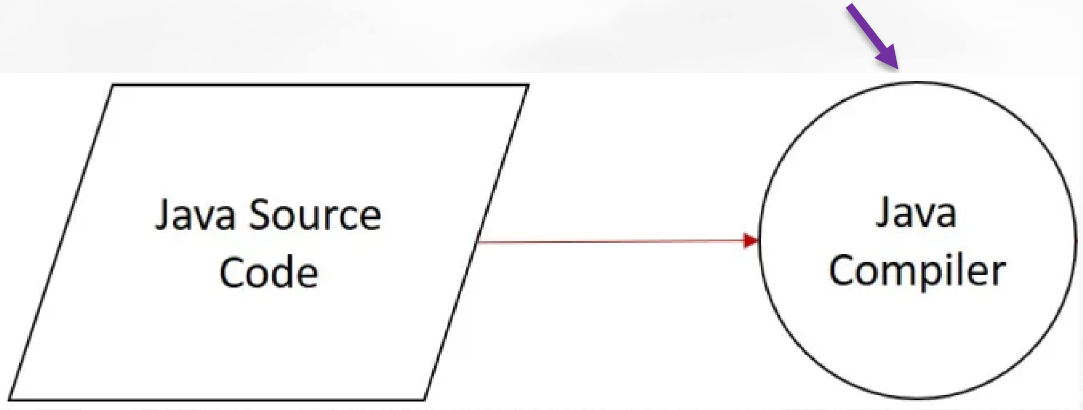
- Download the JDK available at https://docs.oracle.com/en/java/javase/index.html
- Install it
- Create the Java program
- **Compile and run the Java program**

Compile and run the Java program

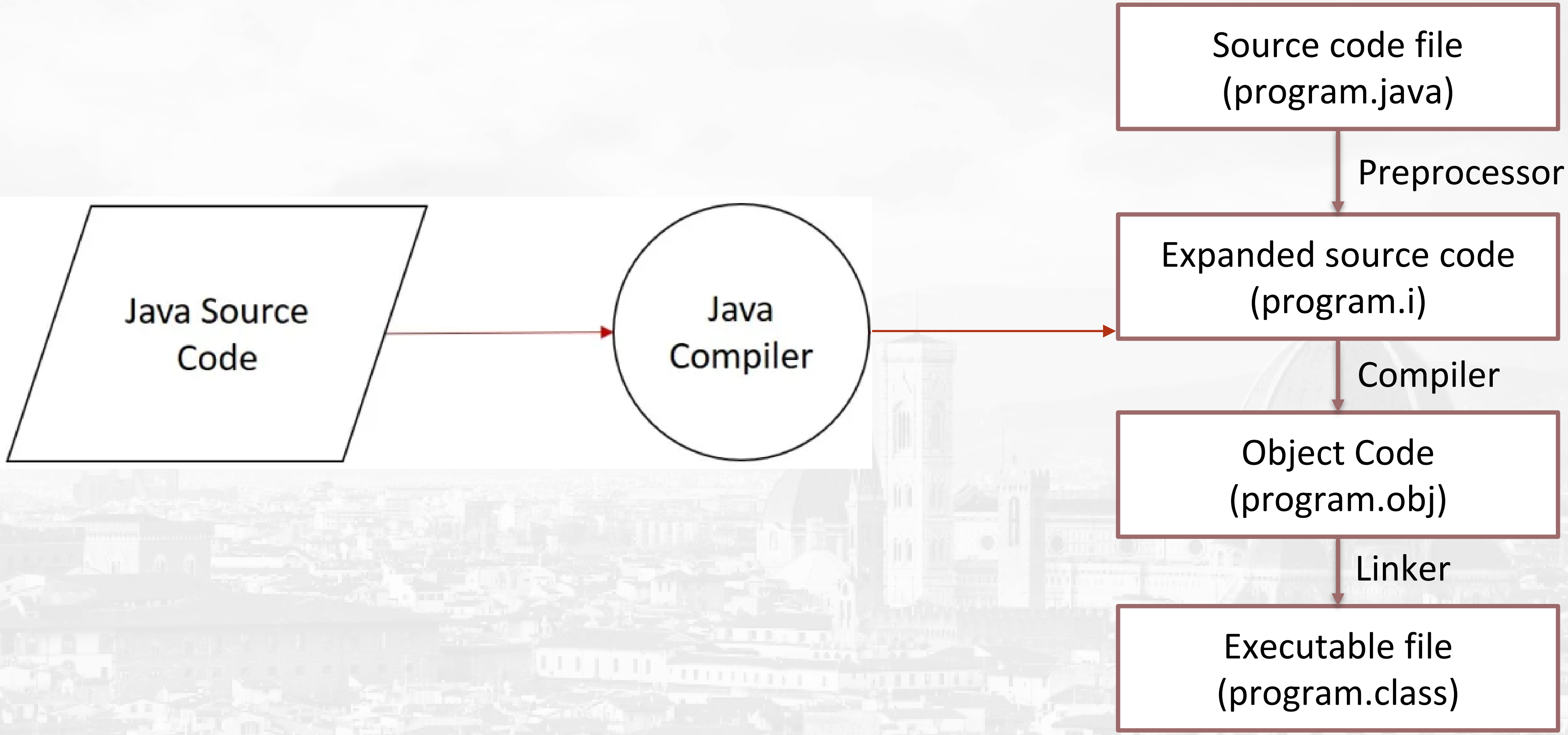
- Java, being a platform-independent programming language, doesn’t work on the one-step compilation.
- It involves a two-step execution
  - Through an OS-independent compiler
  - JVM is custom-built for every operating system.

Compile and run the Java program

The source code of a Java program is compiled to an intermediate binary code called the Bytecode



Compile and run the Java program



Source code file (program.java)

Preprocessor

Expanded source code (program.i)

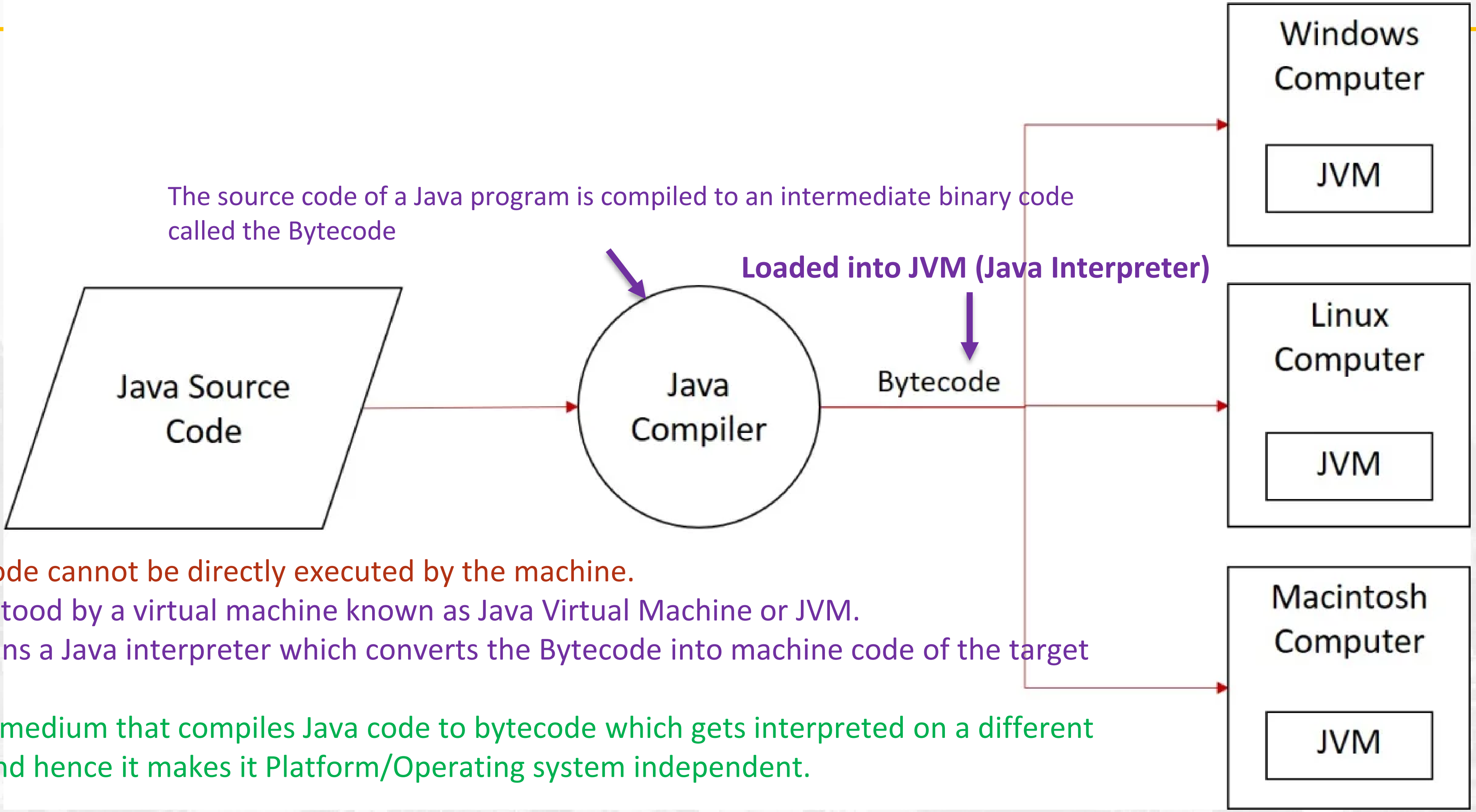
Compiler

Object Code (program.obj)

Linker

Executable file (program.class)

Compile and run the Java program



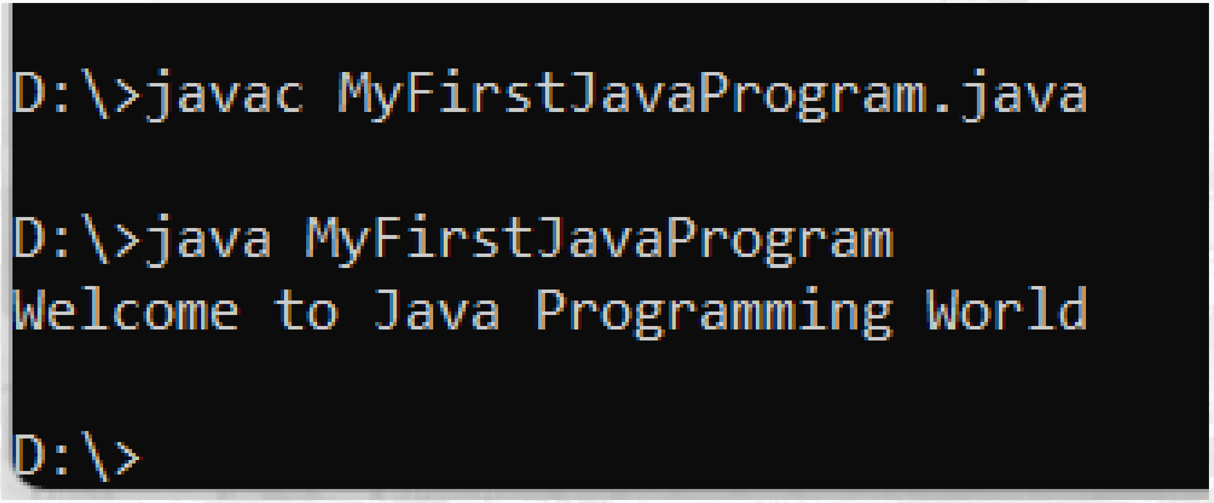
The source code of a Java program is compiled to an intermediate binary code called the Bytecode

Loaded into JVM (Java Interpreter)

- This Bytecode cannot be directly executed by the machine.
- It is understood by a virtual machine known as Java Virtual Machine or JVM.
- JVM contains a Java interpreter which converts the Bytecode into machine code of the target computer.
- JVM is the medium that compiles Java code to bytecode which gets interpreted on a different machine and hence it makes it Platform/Operating system independent.

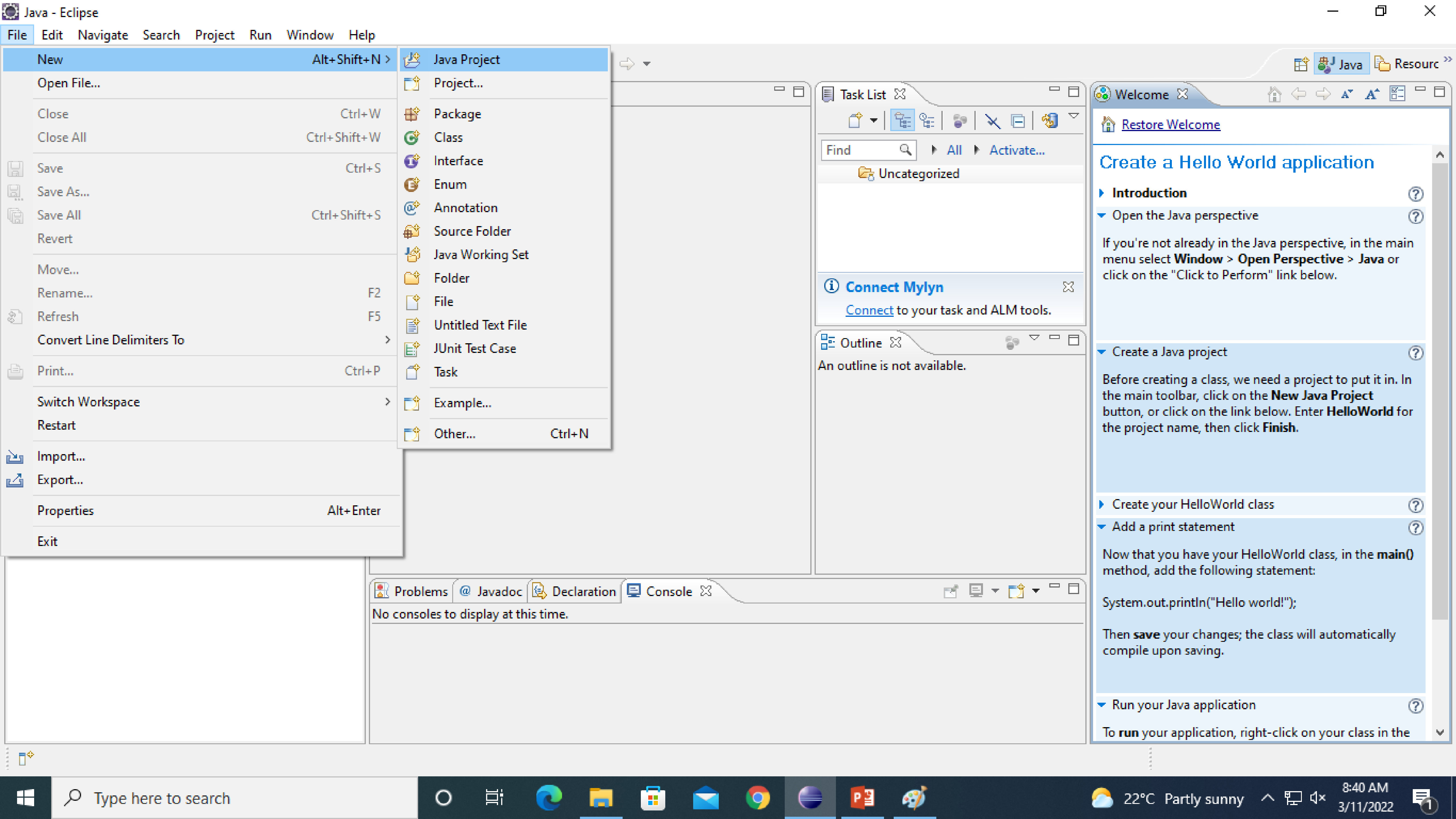
Compile and run the Java program

- To compile
  - Open the command prompt
  - Type the command “javac MyFirstJavaProgram.java”
- To execute
  - java MyFirstJavaProgram



```
D:\>javac MyFirstJavaProgram.java

D:\>java MyFirstJavaProgram
Welcome to Java Programming World

D:\>
```

Example using Eclipse IDE



Connect to your task and ALM tools.

An outline is not available.

No consoles to display at this time.

Create a Hello World application

Introduction

Open the Java perspective

If you're not already in the Java perspective, in the main menu select **Window** > **Open Perspective** > **Java** or click on the "Click to Perform" link below.

Create a Java project

Before creating a class, we need a project to put it in. In the main toolbar, click on the **New Java Project** button, or click on the link below. Enter **HelloWorld** for the project name, then click **Finish**.

Create your HelloWorld class

Add a print statement

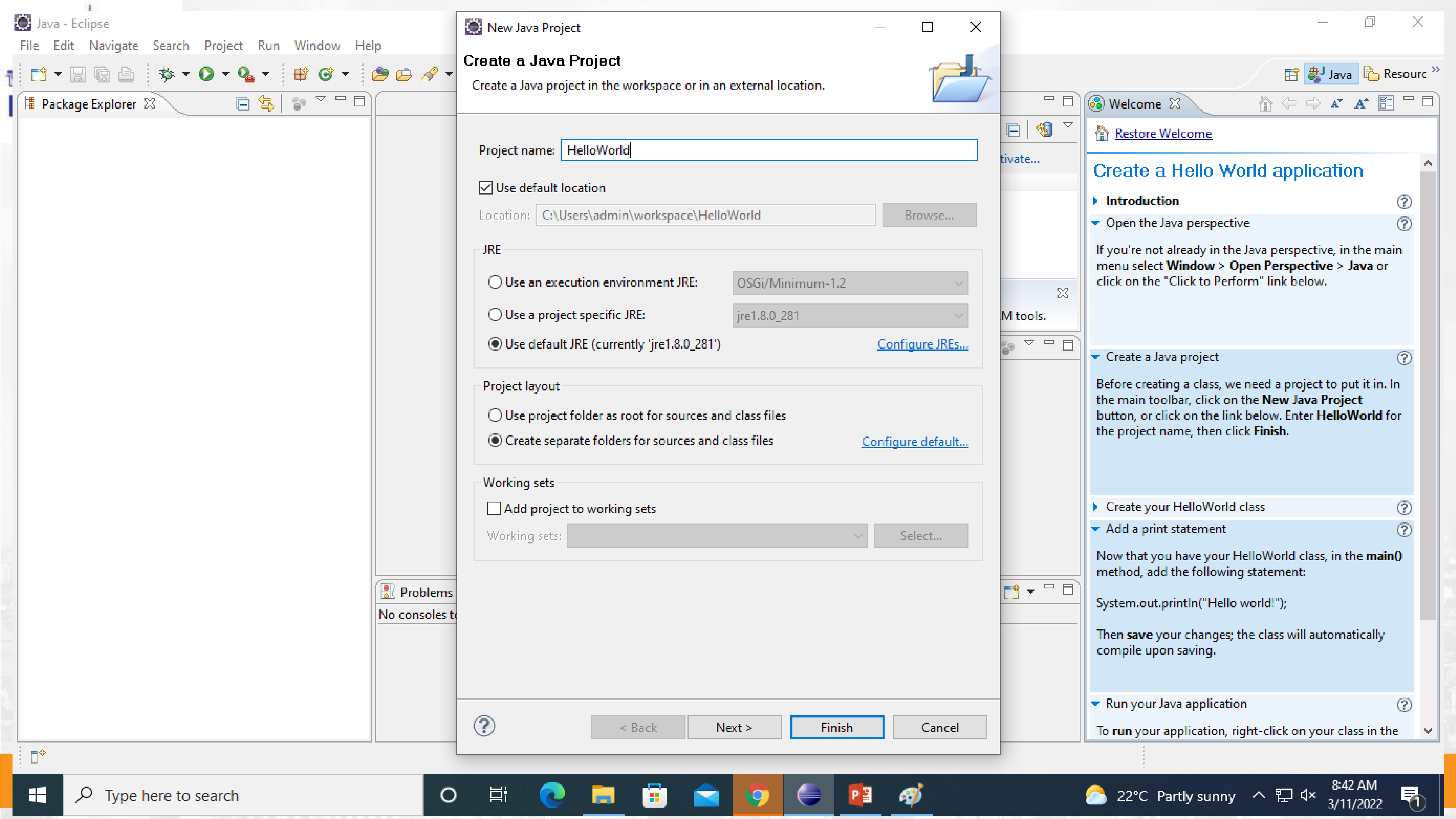
Now that you have your HelloWorld class, in the **main()** method, add the following statement:

```
System.out.println("Hello world!");
```

Then **save** your changes; the class will automatically compile upon saving.

Run your Java application

To **run** your application, right-click on your class in the



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: HelloWorld

☑ Use default location

Location: C:\Users\admin\workspace\HelloWorld

JRE

○ Use an execution environment JRE: OSGi/Minimum-1.2

○ Use a project specific JRE: jre1.8.0_281

● Use default JRE (currently 'jre1.8.0_281') Configure JREs...

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files Configure default...

Working sets

□ Add project to working sets

Working sets:

Create a Hello World application

Introduction

Open the Java perspective

If you're not already in the Java perspective, in the main menu select **Window** > **Open Perspective** > **Java** or click on the "Click to Perform" link below.

Create a Java project

Before creating a class, we need a project to put it in. In the main toolbar, click on the **New Java Project** button, or click on the link below. Enter **HelloWorld** for the project name, then click **Finish.**

Create your HelloWorld class

Add a print statement

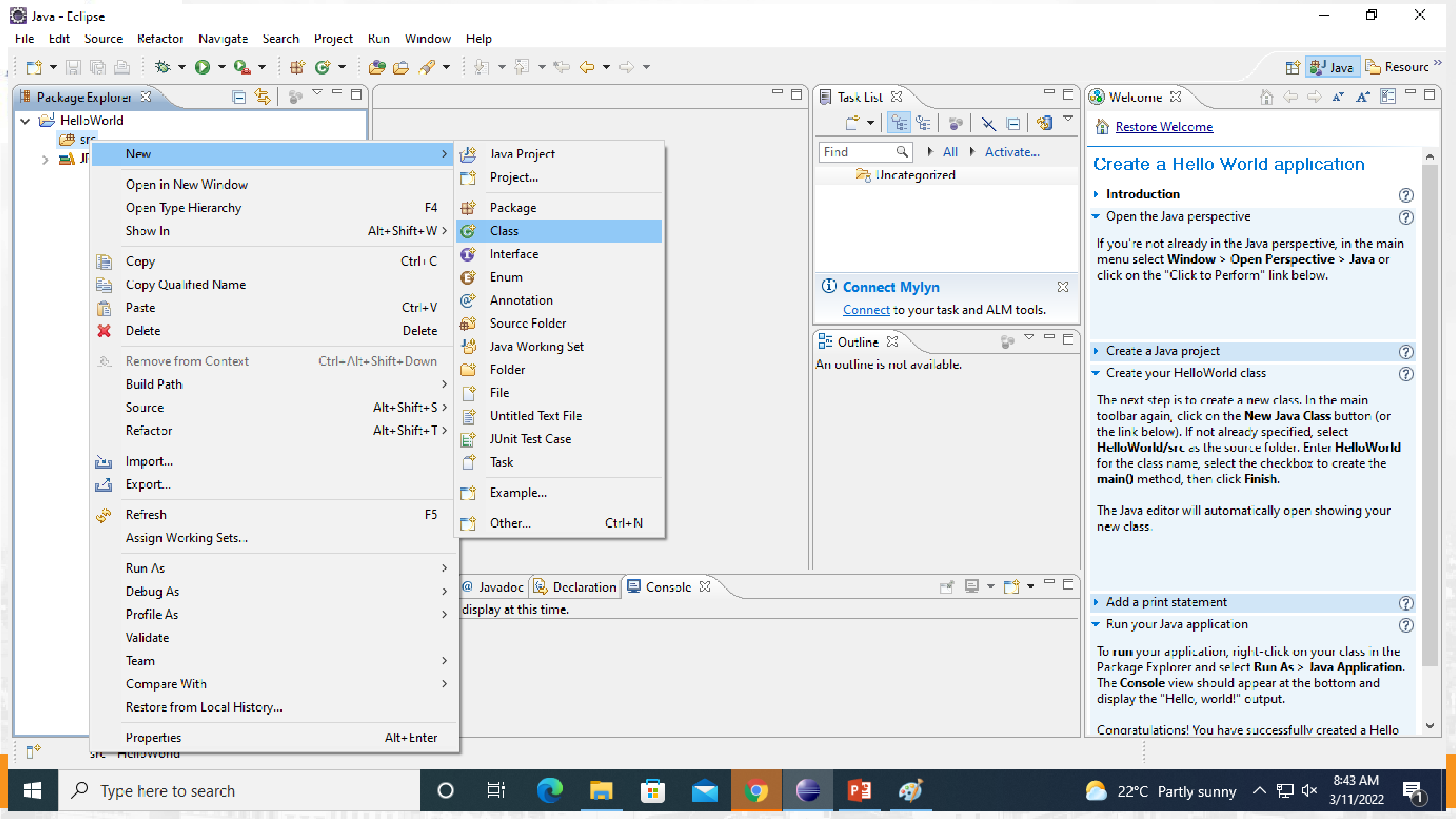
Now that you have your HelloWorld class, in the **main()** method, add the following statement:

```
System.out.println("Hello world!");
```

Then **save** your changes; the class will automatically compile upon saving.

Run your Java application

To **run** your application, right-click on your class in the



Create a Hello World application

Introduction

Open the Java perspective

If you're not already in the Java perspective, in the main menu select **Window** > **Open Perspective** > **Java** or click on the "Click to Perform" link below.

Create a Java project

Create your HelloWorld class

The next step is to create a new class. In the main toolbar again, click on the **New Java Class** button (or the link below). If not already specified, select **HelloWorld/src** as the source folder. Enter **HelloWorld** for the class name, select the checkbox to create the **main()** method, then click **Finish**.

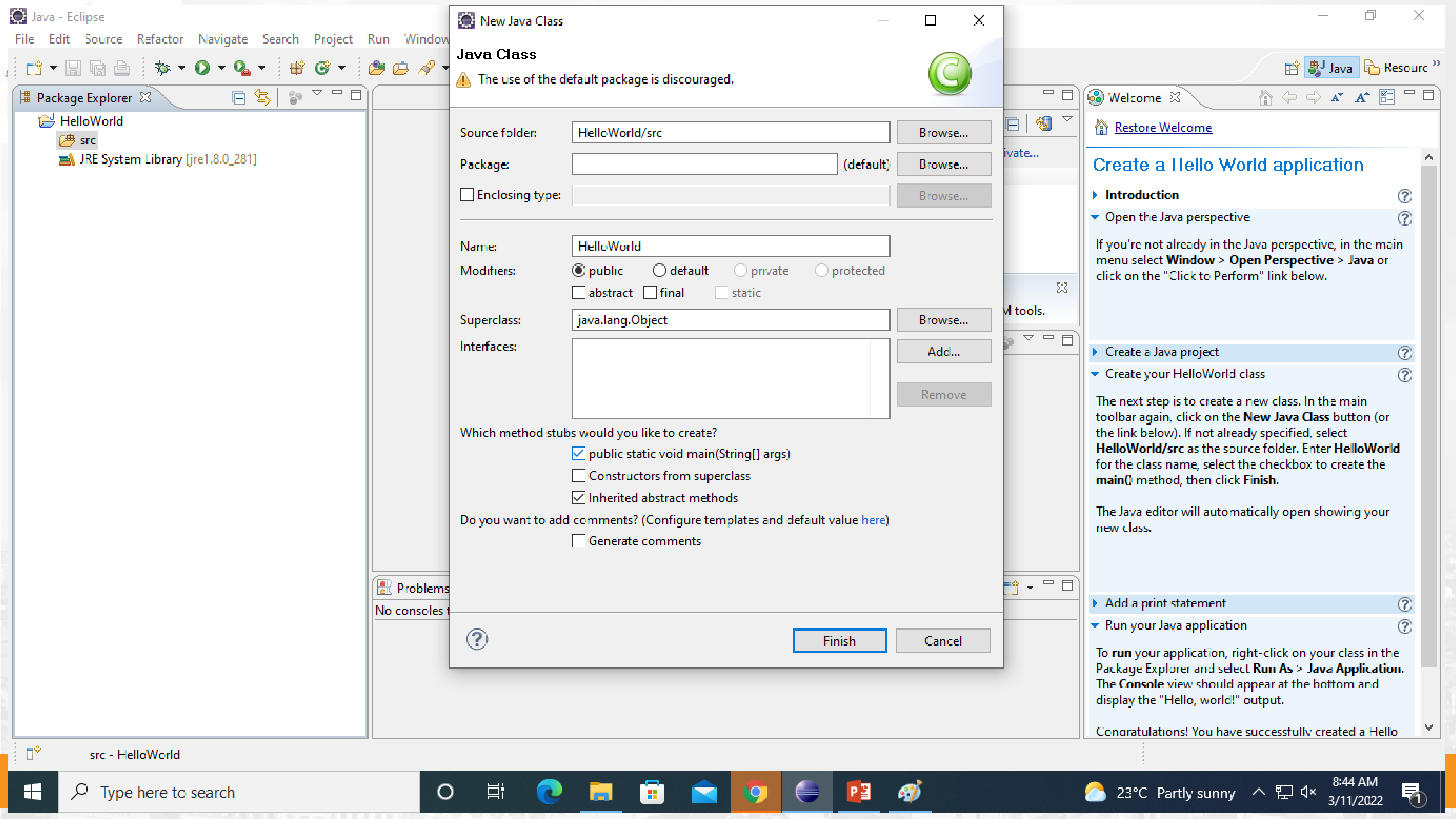
The Java editor will automatically open showing your new class.

Add a print statement

Run your Java application

To **run** your Java application, right-click on your class in the Package Explorer and select **Run As** > **Java Application**. The **Console** view should appear at the bottom and display the "Hello, world!" output.

Congratulations! You have successfully created a Hello



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: HelloWorld/src [Browse...]

Package: (default) [Browse...]

□ Enclosing type: [Browse...]

Name: HelloWorld

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object [Browse...]

Interfaces: [Add...] [Remove]

Which method stubs would you like to create?

☑ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

[Finish] [Cancel]

Create a Hello World application

- Introduction
- Open the Java perspective

If you're not already in the Java perspective, in the main menu select **Window** > **Open Perspective** > **Java** or click on the "Click to Perform" link below.

- Create a Java project
- Create your HelloWorld class

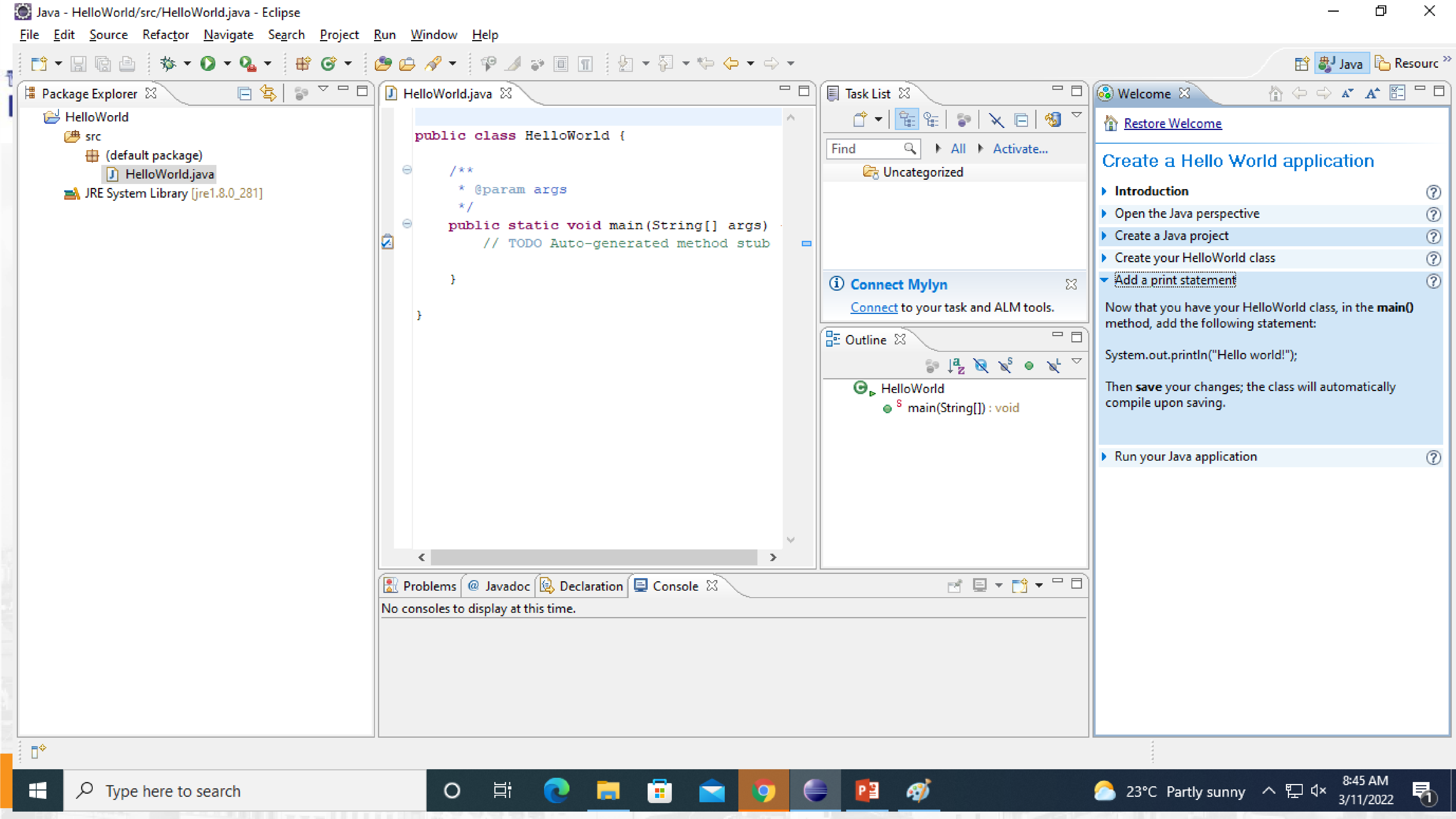
The next step is to create a new class. In the main toolbar again, click on the **New Java Class** button (or the link below). If not already specified, select **HelloWorld/src** as the source folder. Enter **HelloWorld** for the class name, select the checkbox to create the **main()** method, then click **Finish.**

The Java editor will automatically open showing your new class.

- Add a print statement
- Run your Java application

To **run** your application, right-click on your class in the Package Explorer and select **Run As** > **Java Application.** The **Console** view should appear at the bottom and display the "Hello, world!" output.

Congratulations! You have successfully created a Hello



```
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args)
        // TODO Auto-generated method stub

    }

}
```

Connect Mylyn

Connect to your task and ALM tools.

No consoles to display at this time.

Create a Hello World application

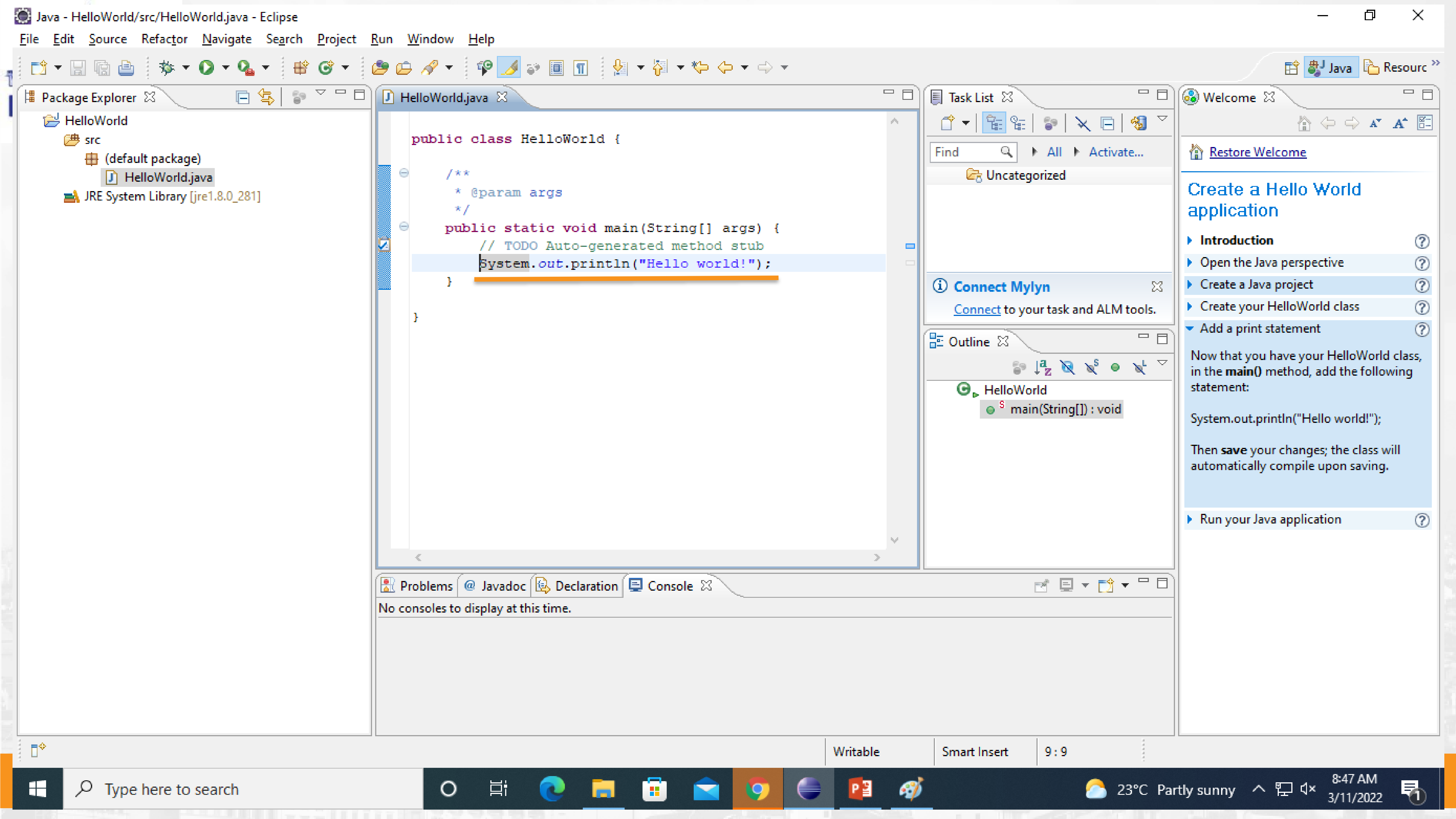
- Introduction
- Open the Java perspective
- Create a Java project
- Create your HelloWorld class
- Add a print statement

Now that you have your HelloWorld class, in the **main()** method, add the following statement:

System.out.println("Hello world!");

Then **save** your changes; the class will automatically compile upon saving.

- Run your Java application



```
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hello world!");
    }

}
```

No consoles to display at this time.

Create a Hello World application

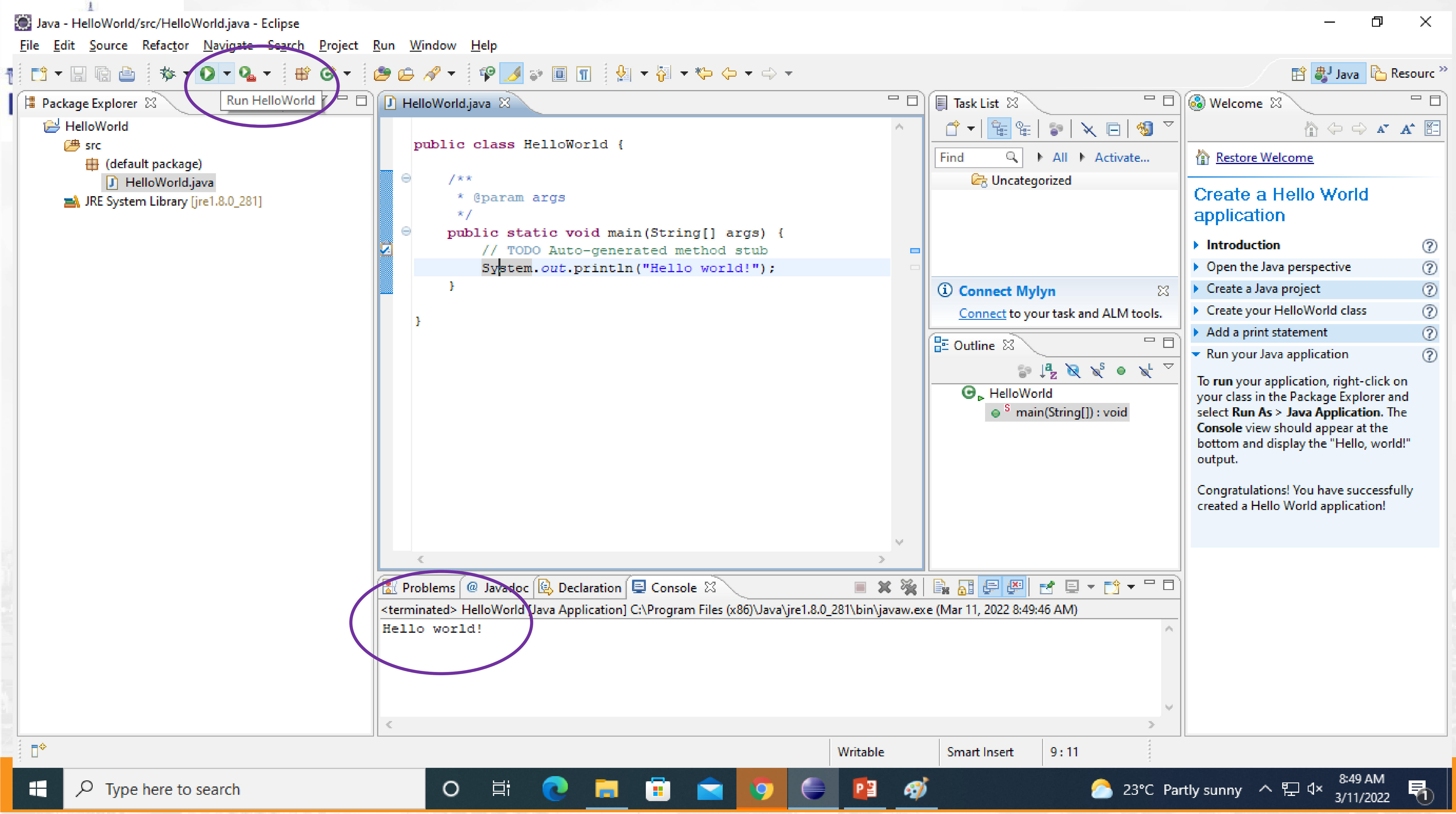
- Introduction
- Open the Java perspective
- Create a Java project
- Create your HelloWorld class
- Add a print statement

Now that you have your HelloWorld class, in the **main()** method, add the following statement:

System.out.println("Hello world!");

Then **save** your changes; the class will automatically compile upon saving.

- Run your Java application



```
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hello world!");
    }

}
```

```
<terminated> HelloWorld [Java Application] C:\Program Files (x86)\Java\jre1.8.0_281\bin\javaw.exe (Mar 11, 2022 8:49:46 AM)
Hello world!
```

Create a Hello World application

- Introduction
- Open the Java perspective
- Create a Java project
- Create your HelloWorld class
- Add a print statement
- Run your Java application

To **run** your application, right-click on your class in the Package Explorer and select **Run As** > **Java Application**. The **Console** view should appear at the bottom and display the "Hello, world!" output.

Congratulations! You have successfully created a Hello World application!

Integrated Development Environment (IDE)

- To write your Java programs, there is a need of text editor. Even more sophisticated IDEs can be used.
  - Netbeans − A Java IDE that is open-source and free which can be downloaded from https://www.netbeans.org/index.html.
  - Eclipse − A Java IDE developed by the eclipse open-source community and can be downloaded from https://www.eclipse.org/.

Java Comments

- Single-line comments start with two forward slashes (//).
- Multi-line comments start with /* and ends with */.

Java Variables

- Variables are containers for storing data values.
- Different types of variables
  - String - stores text, such as "Hello". String values are surrounded by double quotes
  - int - stores integers (whole numbers), without decimals, such as 123 or -123
  - float - stores floating point numbers, with decimals, such as 19.99 or -19.99
  - char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
  - boolean - stores values with two states: true or false
- To create a variable, syntax is –
  - type variableName = value;
  - Example

```
int a = 15;
System.out.println(a);
```

Final Variables

- Add the “final” keyword if you don't want others (or yourself) to overwrite existing values.
- This will declare the variable as "final" or "constant", which means unchangeable.
- Example

```
final int a = 15;
a = 20;      Will give an error
```

Java Primitive Data Types

- Integer Types
  - Byte
    - Stores whole numbers from -128 to 127

```
byte a = 100;
System.out.println(a);
```

  - Short
    - Stores whole numbers from -32768 to 32767

```
short b = 100;
System.out.println(b);
```

  - Int
    - Stores whole numbers from -2147483648 to 2147483647

```
int c = 100;
System.out.println(c);
```

  - Long
    - Stores whole numbers from -9223372036854775808 to 9223372036854775807

```
long d = 100;
System.out.println(d);
```

Java Primitive Data Types

- Floating Point Types
  - Float
    - Stores fractional numbers from 3.4e−038 to 3.4e+038
    - Value should end the value with an "f"

```
float f1 = 5.75f;
System.out.println(f1);
```

```
Output -
5.75
```

  - Double
    - Stores fractional numbers from 1.7e−308 to 1.7e+308
    - Value should end the value with an “d"

```
double f1 = 19.99d;
System.out.println(f1);
```

```
Output -
19.99
```

  - Scientific Numbers
    - A floating point number can also be a scientific number with an "e" to indicate the power of 10

```
float f1 = 35e3f;
double d1 = 12E4d;
System.out.println(f1);
System.out.println(d1);
```

```
Output -
35000.0
120000.0
```

Java Primitive Data Types

- Booleans
  - A boolean data type is declared with the boolean keyword and can only take the values true or false

```
boolean isJavaFun = true;
System.out.println(isJavaFun);
```

```
Output -
true
```

- Characters
  - Stores a single character

```
char var1 = ‘A';
System.out.println(var1);
```

```
Output -
A
```

  - Can use ASCII values to display certain characters

```
char var1 = 65;
System.out.println(var1);
```

```
Output -
A
```

- Strings
  - Stores a sequence of characters.
  - Must be surrounded by double quotes

```
String greeting = "Hello World";
System.out.println(greeting);
```

```
Output -
Hello World
```

Java Non-Primitive Data Types

- Non-primitive data types are called reference types because they refer to objects.
- Examples
  - Strings, Arrays, Classes, Interface, etc.
- The main difference between primitive and non-primitive data types are
  - Primitive types are predefined 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.

Java Type Casting

- Type casting is when you assign a value of one primitive data type to another type.
- In Java, there are tw
  - Widening Casting (a smaller type to a larger type size
    - byte -> short -> c
  - Narrowing Casting er ty size type
    - double -> float ->

```
int a = 9;
double b = a;

System.out.println(a);
System.out.println(b);
```

```
Output –
9
9.0
```

```
double b = 9.78d;
int a = (int) b;

System.out.println(b);
System.out.println(a);
```

```
Output –
9.78
9
```

Java Programming-The General Form of Class

- A class a new (user defined) data type. Once defined, this new type can be used to create objects of that type.
- A class contains three types of items : variable,methods, and constructors.
- **Variable** represent its state.
- **Method** provide the logic that constitutes the behavior defined by a class.
- Collectively, the methods and variables defined within a class are called *members of the class.*
- **Constructors** initialize the state of a new instance of a class.

General Form of Class contd….

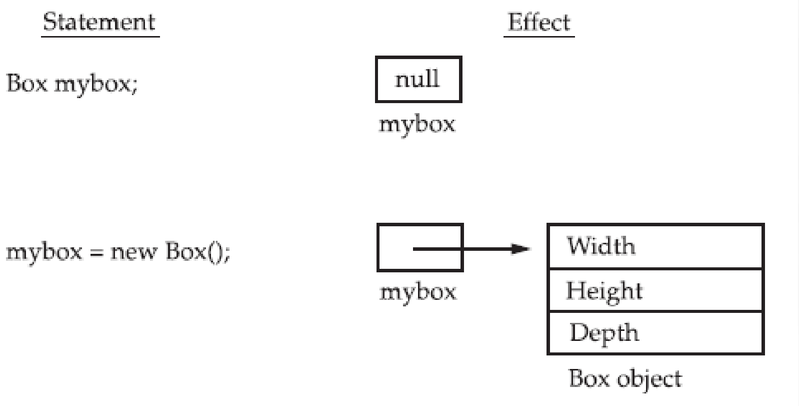
The Simplified form of a class is :

```
class clsName {
//instance variable declarations
type1 varName1;
type2 varName2;
…..
//constructor
clsName(cparams1) {
// body of constructor
}
clsName(cparamsN) {
//body of constructor
}
…
//methods
rtype1 mthName(mparams1) {
//body of method
}
…..
}
}
```

- The keyword class indicates that a class named clsName is being declared. This name must follow the Java naming convention for identifiers. The instance Variables named varName1 through varNameN included using the normal variable declaration syntax. Each variable must be assigned a type shown as

Example of a Simple Class & object

```
class Box {
double width;
double height;
double depth;
}
```



class object creation :-

2-steps for creating objects

Box mybox; // declare reference to object

mybox = new Box(); // // allocate a Box object

OR

Box **mybox** = new Box(); // or create a Box object called mybox in 1-step

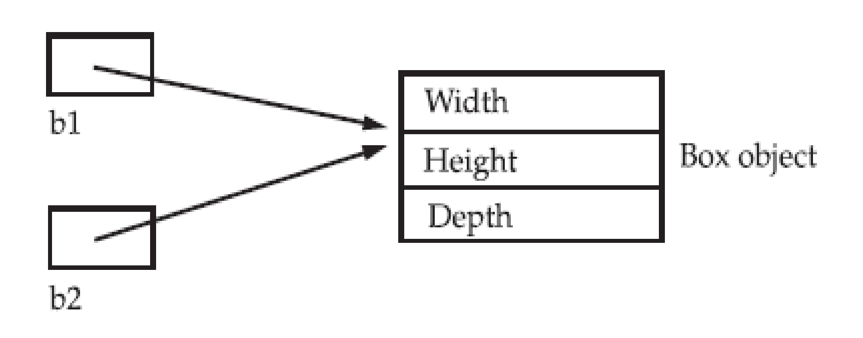
Here **mybox** is a object of class Box.

- Each time you create an instance of a class, you are creating an object that contains its own copy of each instance variable defined by the class.
- Thus, every **Box** object will contain its own copies of the **instance variables width, height, and depth.**
- **To** access these variables, you will use the *dot (.) operator. The dot operator links the name of the* object with the name of an instance variable.

Ex:- **mybox**.width = 100;

Assigning Object Reference Variables

- Box b1 = new Box();
- Box b2 = b1;



Program example for classes and object

```
class Box {
double width;
double height;
double depth;
}
class BoxDemo2 {
public static void main(String args[]) {
Box mybox1 = new Box();
Box mybox2 = new Box();
double vol;
// assign values to mybox1's instance variables
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
```

```
  /* assign different values to mybox2's
  instance variables */
  mybox2.width = 3;
  mybox2.height = 6;
  mybox2.depth = 9;
  // compute volume of first box
  vol = mybox1.width * mybox1.height * mybox1.depth;
  System.out.println("Volume is " + vol);
  // compute volume of second box
  vol = mybox2.width * mybox2.height * mybox2.depth;
  System.out.println("Volume is " + vol);
 }
}
```

Constructors

- Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor.

- A *constructor initializes an object immediately upon creation. It has the same name as the* class in which it resides and is syntactically similar to a method.

- Once defined, the constructor is automatically called immediately after the object is created, before the **new operator completes.**

- **Note :-** They have no return type, not even **void.**

Constructor Example

```
class Box {
double width;
double height;
double depth;
// This is the constructor for Box.
Box() {
System.out.println("Constructing Box");
width = 10;
height = 10;
depth = 10;
}
// compute and return volume
double volume() {
return width * height * depth;
}
}
```

```
class BoxDemo6 {
public static void main(String args[]) {
// declare, allocate, and initialize Box objects
Box mybox1 = new Box();
Box mybox2 = new Box();
double vol;
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

Introducing Methods

This is the general form of a method:

```
type name(parameter-list)
{
// body of method
}
```

Introduction to Access Control

- Encapsulation provides important attribute: *access control*

- Through encapsulation, you can control what parts of a program can access the members of a class.

- By controlling access, you can prevent misuse. For example, allowing access to data only through a well defined set of methods, you can prevent the misuse of that data.

- How a member can be accessed is determined by the *access specifier that modifies its* declaration.

- Java supplies a rich set of access specifiers.

Access Control

- Java’s access specifiers are **public, private, and protected.**
- **Java also defines a default** access level. When no access specifier is used, then by **default** the **member of a class is public** within its own package, but cannot be accessed outside of its package.
- **Protected applies only when inheritance is involved.**
- **The other access specifiers** are described next.

Program to Demonstrate the Difference between Public and Private

```
/* This program demonstrates the difference
      between public and private.
*/
class Test {
                int a; // default access
                public int b; // public access
                 private int c; // private access
  // methods to access c
 void setc(int i) { // set c's value
                          c = i;
                      }
int getc() { // get c's value
   return c;
}
}
```

```
class AccessTest {
public static void main(String args[]) {
               Test ob = new Test();
               // These are OK, a and b may be accessed directly
             ob.a = 10;
            ob.b = 20;
// This is not OK and will cause an error
// ob.c = 100;  // Error!
// You must access c through its methods
ob.setc(100); // OK
System.out.println("a, b, and c: " + ob.a + " " + ob.b + " " + ob.getc());
}
}
```

Adding a Method to the Box Class

```
class Box {
    double width;
    double height;
    double depth;
    // display volume of a box
    void volume() {
        System.out.print("Volume is ");
        System.out.println(width * height * depth);
    }
}
```

Adding a Method to the Box Class contd..

```
class BoxDemo3 {
        public static void main(String arg[]) {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        // assign values to mybox1's instance variables
        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;
        /* assign different values to mybox2's instance variables */
        mybox2.width = 3;
        mybox2.height = 6;
        mybox2.depth = 9;
        // display volume of first box
        mybox1.volume();
        // display volume of second box
        mybox2.volume();
        }
}
```

Returning a Value

```
class Box {
    double width;
    double height;
    double depth;
    // compute and return volume
    double volume() {
    return width * height * depth;
    }
}
```

Program_Example returning a Value

```
class BoxDemo4 {
public static void main(String args[]) {
Box mybox1 = new Box();
Box mybox2 = new Box();
double vol;
// assign values to mybox1's instance variables
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
/* assign different values to mybox2's
instance variables */
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9;
```

Program_Example returning a Value contd…

```
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

Adding a Method that Takes Parameters

```
class Box {
  double width;
  double height;
  double depth;
 // compute and return volume
 double volume() {
    return width * height * depth;
 }
// sets dimensions of box
void setDim(double w, double h, double d) {
  width = w;
  height = h;
  depth = d;
 }
}
```

Adding a Method that Takes Parameters   contd..

```
class BoxDemo5 {
       public static void main(String args[]) {
       Box mybox1 = new Box();
       Box mybox2 = new Box();
      double vol;
      // initialize each box
      mybox1.setDim(10, 20, 15);
      mybox2.setDim(3, 6, 9);
     // get volume of first box
      vol = mybox1.volume();
      System.out.println("Volume is " + vol);
      // get volume of second box
     vol = mybox2.volume();
     System.out.println("Volume is " + vol);
  }       }
```

A Closer Look at Argument Passing

There are two ways that a computer language can pass an argument to a subroutine.

i) **Call by Value :-** This approach copies the *value of an argument into the formal* parameter of the subroutine. Therefore, changes made to the parameter of the subroutine have no effect on the argument.

ii) *Call-by-reference:-* In this approach, a reference to an argument (not the value of the argument) is passed to the parameter.

Inside the subroutine, this reference is used to access the actual argument specified in the call.

This means that changes made to the parameter will affect the argument used to call the subroutine.

A Closer Look at Argument Passing   contd..

- In Java, when you pass a **primitive type** to a method, it is **passed by value**.
- When you **pass an object** to a method, the objects are passed by what is **effectively call-by-reference**.

Pass by value example

```
// Primitive types are passed by value.
class Test {
void meth(int i, int j) {
i *= 2;
j /= 2;
}
}
class CallByValue {
public static void main(String args[]) {
Test ob = new Test();
int a = 15, b = 20;
System.out.println("a and b before call: " +a + " " + b);
ob.meth(a, b);
System.out.println("a and b after call: " +a + " " + b);
}
}
```

The output from this program is shown here:

```
a and b before call: 15 20
a and b after call: 15 20
```

Pass by reference example

```
// Objects are passed by reference.
class Test {
int a, b;
Test(int i, int j) {
a = i;
b = j;
}
// pass an object
void meth(Test o) {
o.a *= 2;
o.b /= 2;
}
}
class CallByRef {
 public static void main(String args[]) {
 Test ob = new Test(15, 20);
 System.out.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);
  ob.meth(ob);
 System.out.println("ob.a and ob.b
   after call: " + ob.a + " " + ob.b);
}
}
```

This program generates the following output:

```
ob.a and ob.b before call: 15 20
ob.a and ob.b after call: 30 10
```

Returning Objects

```
// Returning an object.
class Test {
int a;
Test(int i) {
a = i;
}
Test incrByTen() {
Test temp = new Test(a+10);
return temp;
}
}
class RetOb {
public static void main(String args[]) {
Test ob1 = new Test(2);
Test ob2;
ob2 = ob1.incrByTen();
System.out.println("ob1.a: " + ob1.a);
System.out.println("ob2.a: " + ob2.a);
ob2 = ob2.incrByTen();
System.out.println("ob2.a after second increase: “ + ob2.a);
}
}
```

The output generated by this program is shown here:

```
ob1.a: 2
ob2.a: 12
ob2.a after second increase: 22
```

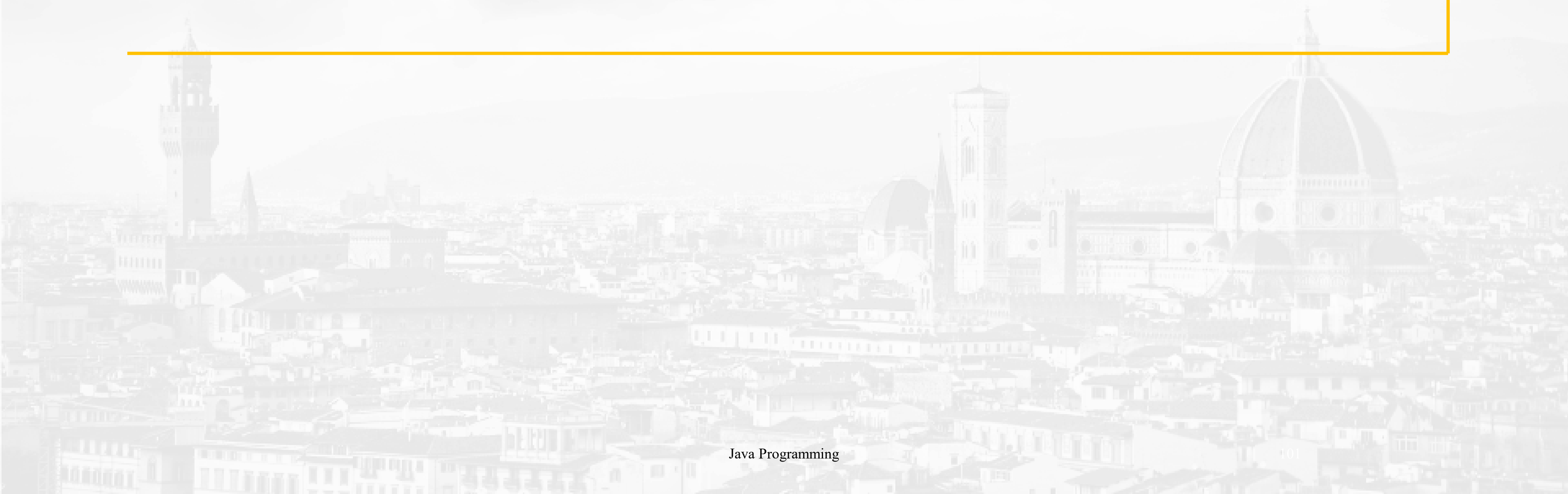
Practice Assignments

- Write a program that asks the user to enter their name in lowercase and then capitalizes the first letter of each word of their name.
- Write a program that asks the user to enter a word and determines whether the word is a palindrome or not. A palindrome is a word that reads the same backwards as forwards.
- Write a program to find the factorial of a given number.
- Write a program that takes ‘a’ an integer as input and displays True if ’a’ is a prime product and False otherwise.
- Write a program that displays the output as below –

```
        *
    *   *   *
*   *   *   *   *
```

FAQs

- What are classes?
- Explain : Array of Objects
- Write couple of examples/applications suitable to use OOP concepts specially use of classes, objects and constructors.



Thank You!!