2023

Java Standard Edition (Java SE) Programming

LEARNING ONE OF THE MOST POPULAR AND WIDELY USED PROGRAMMING LANGUAGE IN THE WORLD

COMPILED BY JOHNREYGOH

Contents

The JAVA programming language

The Java programming language, which is offered by Oracle, is more difficult to learn than some other languages such as VB and PHP, but it's a great starting place for several reasons. One advantage of learning Java is that you can use it across a variety of operating systems and computing environments. Java programs can be desktop software, web applications, web servers, Android apps, and more, running on Windows, Mac, Linux, and other operating systems. This versatility is referenced by the ambitious early Java slogan "Write once, run anywhere."

Java is unusual because it requires both a compiler and an interpreter. The compiler converts the statements that make up the program into bytecode. Once this bytecode has been created successfully, it can be run by an interpreter called the Java Virtual Machine.
The Java Virtual Machine, also called a JVM, is the thing that makes it possible for the same Java program to run without modification on different operating systems and different kinds of computing devices. The virtual machine turns bytecode into instructions that a particular device's operating system can execute.

To start writing Java programs, you must have a Java programming tool. Several such programs are available for Java, including the simple Java Development Kit and the more sophisticated Eclipse, IntelliJ IDEA, and NetBeans. The latter three tools are each an integrated development environment (IDE), a powerful tool used by professional programmers to get work done. Whenever Oracle releases a new version of Java, the first tool that supports it is the Java Development Kit (JDK).

As of March 2019, Java 8 is supported; and both Java 8 and 11 as Long Term Support (LTS) versions. Major release versions of Java, along with their release dates:

JDK 1.0 (January 23, 1996)[40]
JDK 1.1 (February 19, 1997)
J2SE 1.2 (December 8, 1998)
J2SE 1.3 (May 8, 2000)
J2SE 1.4 (February 6, 2002)
J2SE 5.0 (September 30, 2004)
Java SE 6 (December 11, 2006)
Java SE 7 (July 28, 2011)
Java SE 8 (March 18, 2014)
Java SE 9 (September 21, 2017)
Java SE 10 (March 20, 2018)

Java SE 11 (September 25, 2018)[41]
Java SE 12 (March 19, 2019)

Sun has defined and supports four editions of Java targeting different application environments and segmented many of its APIs so that they belong to one of the platforms. The platforms are:

Java Card for smart-cards.
Java Platform, Micro Edition (Java ME) – targeting environments with limited resources.
Java Platform, Standard Edition (Java SE) – targeting workstation environments.
Java Platform, Enterprise Edition (Java EE) – targeting large distributed enterprise or Internet environments.

Requirements for JAVA application development

To start writing Java programs, you must have a Java programming tool. Several such programs are available for Java, including the simple Java Development Kit and the more sophisticated Eclipse, IntelliJ IDEA, and NetBeans. The latter three tools are each an integrated development environment (IDE), a powerful tool used by professional programmers to get work done. Whenever Oracle releases a new version of Java, the first tool that supports it is the Java Development Kit (JDK).

Download JDK (which includes JRE and JVM) or just the JRE here:
https://www.oracle.com/technetwork/java/javase/downloads/index.html

Download Eclipse IDE here:
https://www.eclipse.org/downloads/

Download NetBeans IDE here:
https://netbeans.org/downloads/

Classes, Objects and Methods

To create a class, use the keyword class:

```
public class MyClass {
  int x = 5;
}
```

To create an object of MyClass, define a method, specify the class name, followed by the object name, and use the keyword new:

```
public class MyClass {
  int x = 5;
```

```
  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Declaring variables and data types

```
// This is a comment
System.out.println("Hello World");
```

```
/* The code below will print the words Hello World
to the screen, and it is amazing */
System.out.println("Hello World");
```

Data types are divided into two groups:

- Primitive data types - includes byte, short, int, long, float, double, boolean and char
- Non-primitive data types - such as String, Arrays and Classes

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Non-Primitive Data Types

Non-primitive data types are called reference types because they refer to objects.

- 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.

Sample variable declarations:

```
int myNum = 5;                  // Integer (whole number)
float myFloatNum = 5.99f;       // Floating point number
char myLetter = 'D';            // Character
boolean myBool = true;          // Boolean
String myText = "Hello";        // String
```

Arithmetic and relational operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value from another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Obtaining input from the console

```
//using the Scanner object
System.out.print("Enter a string : ");
Scanner scanner = new Scanner(System. in);
String inputString = scanner. nextLine();
System.out.println("String read from console is : \n"+inputString);
```

Converting Data Types

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

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

```
//integer to string
int num = 200;
String num1 = num.toString();
String num2 = Integer.toString(num);
```

```
//String to int or double
String num = "200";
int num1 = Integer.parseInt(num);
double num2 = Double.parseDouble(num);
```

String functions

The java.lang.String class provides many useful methods to perform operations on sequence of char values. Here are commonly used String class functions

Method	Description
char charAt(int index)	returns char value for the particular index
int length()	returns string length
static String format(String format, Object... args)	returns a formatted string.
String substring(int beginIndex)	returns substring for given begin index.

String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
boolean equals(Object another)	checks the equality of string with the given object.
boolean isEmpty()	checks if string is empty.
String concat(String str)	concatenates the specified string.
String replace(char old, char new)	replaces all occurrences of the specified char value.
static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
String[] split(String regex)	returns a split string matching regex.
String[] split(String regex, int limit)	returns a split string matching regex and limit.
String toLowerCase()	returns a string in lowercase.
String toUpperCase()	returns a string in uppercase.
String trim()	removes beginning and ending spaces of this string.

Using Dates and Time

```
String mdate = new SimpleDateFormat("dd").format(new Date()).toString();
String mdayname = new SimpleDateFormat("EEEE").format(new Date()).toString();
String mmonth = new SimpleDateFormat("MM").format(new Date()).toString();
String mmonthname = new SimpleDateFormat("MMMM").format(new Date()).toString();
String myear = new SimpleDateFormat("yyyy").format(new Date()).toString();
String m12hour = new SimpleDateFormat("hh").format(new Date()).toString();
String m24hour = new SimpleDateFormat("HH").format(new Date()).toString();
String mmin = new SimpleDateFormat("mm").format(new Date()).toString();
String msec = new SimpleDateFormat("ss").format(new Date()).toString();
String mampm = new SimpleDateFormat("a").format(new Date()).toString();

String c = new SimpleDateFormat("MMMM dd, yyyy(EEE) hh:mm:ss a").format(new Date()).toString();

System.out.print(c);
```

Java Arrays

```
//create an array, then display an item and count the array
String[] reindeerNames = { "Dasher", "Dancer", "Prancer", "Vixen", "Comet", "Cupid", "Donder", "Blitzen", "Rudolph" };
System.out.println("First reindeer mentioned:" + reindeerNames[0]);
System.out.println("There are " + reindeerNames.length + " reindeer.");
```

```
//loop though an array using for-each
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

Conditional control structures

```
//using if conditions
int elephantWeight = 900;
int elephantTotal = 13;
int cleaningExpense = 200;
if (elephantWeight > 780) {
  System.out.println("Elephant too big for tightrope act");
}

if (elephantTotal > 12) {
  cleaningExpense = cleaningExpense + 150;
}
```

```
//using if-else
if (answer == correctAnswer) {
 score += 10;
 System.out.println("That's right. You get 10 points");
} else {
 score -= 5;
 System.out.println("Sorry, that's wrong. You lose 5 points");
}
```

```
//using if-else if
if (grade == 'A') {
  System.out.println("You got an A. Awesome!");
} else if (grade == 'B') {
  System.out.println("You got a B. Beautiful!");
} else if (grade == 'C') {
  System.out.println("You got a C. Concerning!");
} else {
  System.out.println("You got an F. You'll do well in Congress!");
}
```

```
//using switch statements
char grade = 'B';
switch (grade) {
  case 'A':
    System.out.println("You got an A. Awesome!");
    break;
```

```
  case 'B':
    System.out.println("You got a B. Beautiful!");
    break;

  case 'C':
    System.out.println("You got a C. Concerning!");
    break;

  default:
    System.out.println("You got an F. You'll do well in Congress!");
}
```

Loop Structures

```
//for loops
for (int dex = 0; dex < 1000; dex++) {
  if (dex % 12 == 0) {
    System.out.println("#: " + dex);
  }
}
```

```
//while loops
int limit = 5;
int count = 1;
while (count < limit) {
  System.out.println("Pork is not a verb");
  count++;
}
```

```
//do-while loops
int limit = 5;
int count = 1;
do {
  System.out.println("I am not allergic to long division");
  count++;
} while (count < limit);
```

```
//exit a loop
int index = 0;
while (index <= 1000) {
  index = index + 5;
  if (index == 400) {
```

```
    break;
  }
}
```

Creating Classes and Custom Methods

```
//creating and using custom Methods
public class MyClass {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

```
//methods with return values
public class MyClass {
  static int myMethod(int x) {
    return 5 + x;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
// Outputs 8 (5 + 3)
```

```
//methods with multiple parameters
public class MyClass {
  static int myMethod(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    int z = myMethod(5, 3);
```

```
    System.out.println(z);
  }
}
// Outputs 8 (5 + 3)
```

Class constructors

Java class constructors are special methods used to initialize objects. When you create a new object, the constructor method is called automatically, and it can set initial values for the object's attributes or perform other setup tasks.

Basic Concepts

- Constructor Name: The constructor must have the same name as the class in which it resides.
- No Return Type: Constructors do not have a return type, not even void.
- Automatic Call: A constructor is called automatically when a new instance of an object is created.

Types of Constructors

- No-Argument Constructor: A constructor that does not take any parameters. If you don't define any constructor in your class, the Java compiler inserts a default no-argument constructor.
- Parameterized Constructor: A constructor that takes parameters. Use this to initialize objects with specific values.

Rules for Constructors

- You can have more than one constructor in a class as long as their parameter signatures are different. This is known as constructor overloading.
- Constructors can call other constructors using this() and super() for current class constructors and parent class constructors, respectively. Note that these calls must be the first statement in a constructor.

Examples

No-Argument Constructor

```
public class MyClass {
   int number;

   // No-argument constructor
   public MyClass() {
      number = 5; // Default value
   }
}
```

Parameterized Constructor

```
public class Vehicle {
   String model;
   int year;
```

```
    // Parameterized constructor
    public Vehicle(String model, int year) {
        this.model = model;
        this.year = year;
    }
}
```

Creating Objects with Constructors

When you create a new object, you call its constructor:

```
MyClass myObj = new MyClass(); // Calls no-argument constructor
Vehicle myCar = new Vehicle("Toyota", 2020); // Calls parameterized constructor
```

Encapsulation

Encapsulation is a fundamental concept in object-oriented programming (OOP), including Java. It's about bundling the data (attributes) and methods (functions) that operate on the data into a single unit or class and restricting access to some of the object's components. This concept is also known as data hiding.

Key Points of Encapsulation

- Data Hiding: Encapsulation allows you to hide the internal state of the object from the outside. This is usually achieved by making the attributes private and providing public methods to access and modify them, known as getters and setters.
- Increased Flexibility and Maintenance: By encapsulating the data, you can change the internal implementation of the class without affecting the classes that use it.
- Control: Encapsulation gives you control over the data by allowing you to set constraints or conditions on the data being set.

How to Implement Encapsulation in Java

- ✓ Declare Class Variables as Private: This restricts direct access from outside the class.
- ✓ Provide Public Getter and Setter Methods: These are methods that allow outside classes to access and modify the private variables.

Example

Consider a class `Person` that encapsulates the attributes `name` and `age`.

```
public class Person {
    // Private variables, hidden from other classes
    private String name;
    private int age;

    // Constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
```

```
    // Getter method for name
    public String getName() {
        return name;
    }

    // Setter method for name
    public void setName(String name) {
        this.name = name;
    }

    // Getter method for age
    public int getAge() {
        return age;
    }

    // Setter method for age
    public void setAge(int age) {
        this.age = age;
    }
}
```

Usage

```
public class TestEncapsulation {
    public static void main(String[] args) {
        Person person = new Person("John", 25);

        // Accessing the Person name and age via getters
        System.out.println("Name: " + person.getName() + ", Age: " + person.getAge());

        // Modifying the Person's age via setter
        person.setAge(26);

        // Accessing the updated age
        System.out.println("Updated Age: " + person.getAge());
    }
}
```

Benefits of Encapsulation

- Control: The main advantage of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code. With encapsulation, we can change one part of the code easily without affecting other parts.
- Ease of Use and Maintenance: Encapsulation also makes the code more manageable by separating the "what" from the "how".

- Increased Security: It protects an object's integrity by preventing outsiders from making unauthorized changes.

Inheritance

Inheritance is a cornerstone of object-oriented programming (OOP), allowing one class to inherit fields and methods from another. It's a mechanism for code reuse and establishing a relationship between classes. In Java, inheritance is implemented using the *extends* keyword.

Key Concepts of Inheritance

- Superclass (Parent Class): The class whose features are inherited.
- Subclass (Child Class): The class that inherits the features from the superclass. It can also have its own additional features.
- ***extends*** Keyword: Used by a subclass to inherit from a superclass.
- Method Overriding: When a method in a subclass has the same name, return type, and parameters as a method in the superclass, the method in the subclass is said to override the method in the superclass.
- ***super*** Keyword: Used within a subclass method to call a superclass's method.
- Constructor Chaining: Through the use of super(), constructors in a subclass can call constructors in the superclass.
- Implementing Inheritance in Java

Example

Suppose we have a Vehicle class (superclass) and we want to create a Car class (subclass) that inherits from it.

```
// Superclass
public class Vehicle {
   private String make;
   private int year;

   public Vehicle(String make, int year) {
      this.make = make;
      this.year = year;
   }

   public void displayInfo() {
      System.out.println("Make: " + make + ", Year: " + year);
   }
}
```

```
// Subclass
public class Car extends Vehicle {
   private int doors;

   public Car(String make, int year, int doors) {
      super(make, year); // Call to superclass's constructor
      this.doors = doors;
```

```
    }

    // Overriding the displayInfo method
    @Override
    public void displayInfo() {
        super.displayInfo(); // Calls the superclass method
        System.out.println("Doors: " + doors);
    }
}
```

Using Inheritance

```
public class TestInheritance {
    public static void main(String[] args) {
        Car car = new Car("Toyota", 2020, 4);
        car.displayInfo(); // Displays: Make: Toyota, Year: 2020, Doors: 4
    }
}
```

Key Points to Remember

- A subclass inherits all public and protected members (fields, methods, nested classes) from its superclass. Private members are not inherited directly, but can be accessed through public or protected methods.
- Java does not support multiple inheritance with classes. A class cannot extend more than one class. However, a class can implement multiple interfaces, which is Java's way to support multiple inheritance in terms of behavior.
- The Object class, present in the java.lang package, is the root of the class hierarchy. Every class has Object as a superclass (either directly or indirectly) if it doesn't extend another class.

Polymorphism

Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different classes to be treated as objects of a common superclass. It's derived from the Greek words "poly" (many) and "morph" (form), signifying the ability to take on many forms.

In Java, polymorphism manifests mainly in two ways: method overriding (also known as runtime polymorphism) and method overloading (compile-time polymorphism).

Method Overriding (Runtime Polymorphism)

Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. This allows a subclass to inherit the method from its superclass but also to modify or replace the superclass's method, depending on the subclass's needs.

Example of Method Overriding

```
class Animal {
    void speak() {
```

```
    System.out.println("This animal speaks in its own special way.");
  }
}

class Dog extends Animal {
  // Overriding the speak method
  @Override
  void speak() {
    System.out.println("The dog barks.");
  }
}

class Cat extends Animal {
  // Overriding the speak method
  @Override
  void speak() {
    System.out.println("The cat meows.");
  }
}
```

Usage

```
Animal myDog = new Dog();
Animal myCat = new Cat();

myDog.speak(); // Outputs: The dog barks.
myCat.speak(); // Outputs: The cat meows.
```

Method Overloading (Compile-Time Polymorphism)

Method overloading occurs within a single class when two or more methods in the same class have the same name but differ in their parameter lists (either in the number of parameters, the type of parameters, or both). Method overloading allows a class to offer several ways to perform an operation or to provide convenience to the class user.

Example of Method Overloading

```
class MathOperation {
  // Method to add two integers
  int add(int a, int b) {
    return a + b;
  }

  // Overloaded method to add three integers
  int add(int a, int b, int c) {
    return a + b + c;
  }
```

```
}
```

Usage

```
MathOperation operation = new MathOperation();
System.out.println(operation.add(5, 7)); // Outputs: 12
System.out.println(operation.add(5, 7, 10)); // Outputs: 22
```

Key Points to Remember

- Polymorphism enables Java objects to take on multiple forms.
- Method Overriding (Runtime Polymorphism) allows a subclass to offer a specific implementation of a method already provided by one of its superclasses.
- Method Overloading (Compile-Time Polymorphism) allows a class to have more than one method having the same name, if their parameter lists are different.
- Polymorphism enhances flexibility and integration by allowing objects of different subclasses to be treated as objects of a superclass.

Abstraction

Abstraction is a core principle in object-oriented programming (OOP) that helps in reducing complexity and increasing efficiency by allowing programmers to focus on the essential qualities of an object rather than its specific implementation. In Java, abstraction can be achieved in two ways: abstract classes and interfaces.

Abstract Classes

An abstract class in Java is a class that cannot be instantiated and may contain abstract methods, which are methods without implementations. Abstract classes are declared with the abstract keyword. The purpose of an abstract class is to provide a base for subclasses to extend and implement the abstract methods.

Key Points:

- If a class includes abstract methods, then the class itself must be declared abstract.
- An abstract class can have both abstract and non-abstract (concrete) methods.
- Subclasses of an abstract class must implement all abstract methods, or they must be declared abstract themselves.

Example:

```
abstract class Animal {
   // Abstract method
   abstract void makeSound();

   // Concrete method
   void breathe() {
      System.out.println("Animal is breathing.");
   }
}
```

```
class Dog extends Animal {
   // Implementing the abstract method
   void makeSound() {
      System.out.println("Bark");
   }
}
```

Interfaces

An interface in Java is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces cannot contain instance fields. The methods in interfaces are abstract by default.

Key Points:

- A class implements an interface, inheriting the abstract methods of the interface.
- A class can implement multiple interfaces, supporting the concept of multiple inheritance.
- Interfaces specify what a class must do but not how it does it.
- From Java 8 onwards, interfaces can also contain default and static methods with implementations.

Example:

```
interface Animal {
   void makeSound(); // Abstract method
}

class Cat implements Animal {
   // Implementing the abstract method from the interface
   public void makeSound() {
      System.out.println("Meow");
   }
}
```

Abstract Classes vs. Interfaces

- Purpose: Abstract classes are used when subclasses share a common structure or behavior. Interfaces are used to define a contract for what a class can do, without specifying how it does it.
- Implementation inheritance vs. type inheritance: Abstract classes provide implementation inheritance (defining and inheriting behavior), whereas interfaces specify type inheritance (defining a contract without enforcing a particular implementation).
- Multiple inheritance: Java does not support multiple inheritance with classes, but by implementing multiple interfaces, a class can inherit from multiple types.

Usage

- ✓ Use abstract classes when you want to share code among several closely related classes.
- ✓ Use interfaces when you want to specify that different classes should comply with a particular contract or when you want to take advantage of multiple inheritance.

Exception Handling

Exception Handling Syntax

Try-Catch

```
try {
   // Code that might throw an exception
} catch (ExceptionType name) {
   // Code to handle the exception
}
```

Try-Catch-Finally

```
try {
   // Code that might throw an exception
} catch (ExceptionType name) {
   // Code to handle the exception
} finally {
   // Code that is always executed
}
```

Throwing an Exception

```
throw new ExceptionType("Error message");
```

Custom Exceptions

You can create your own exception types by extending the Exception class (for checked exceptions) or the RuntimeException class (for unchecked exceptions).

```
class MyCustomException extends Exception {
   public MyCustomException(String message) {
      super(message);
   }
}
```

Examples

Basic Try-Catch

```
try {
   int division = 10 / 0; // This will cause a divide-by-zero exception
} catch (ArithmeticException e) {
   System.out.println("ArithmeticException => " + e.getMessage());
}
```

Try-Catch-Finally

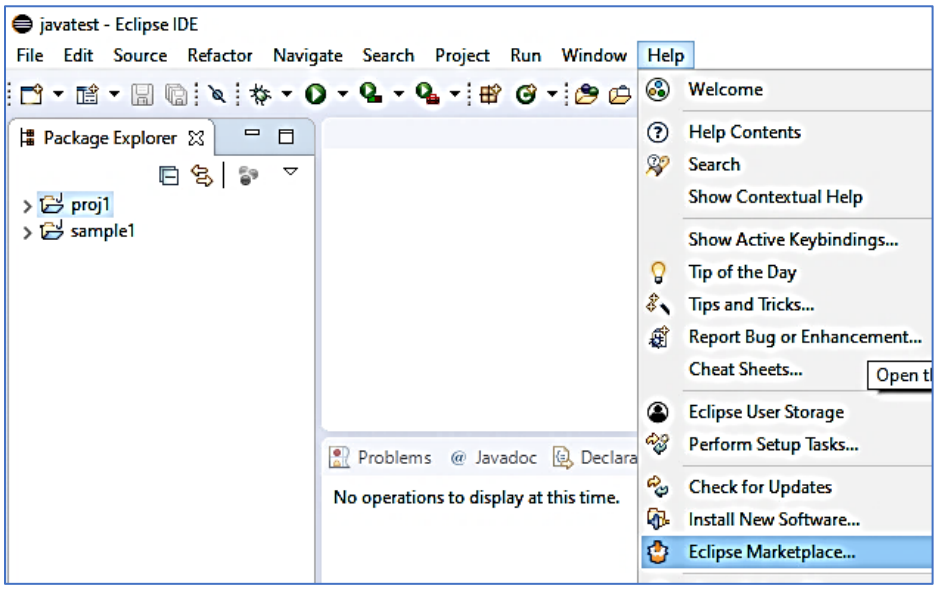
```
try {
   // risky operations
} catch (SpecificException ex) {
   // handling
} finally {
   // cleanup code, always executed
}
```

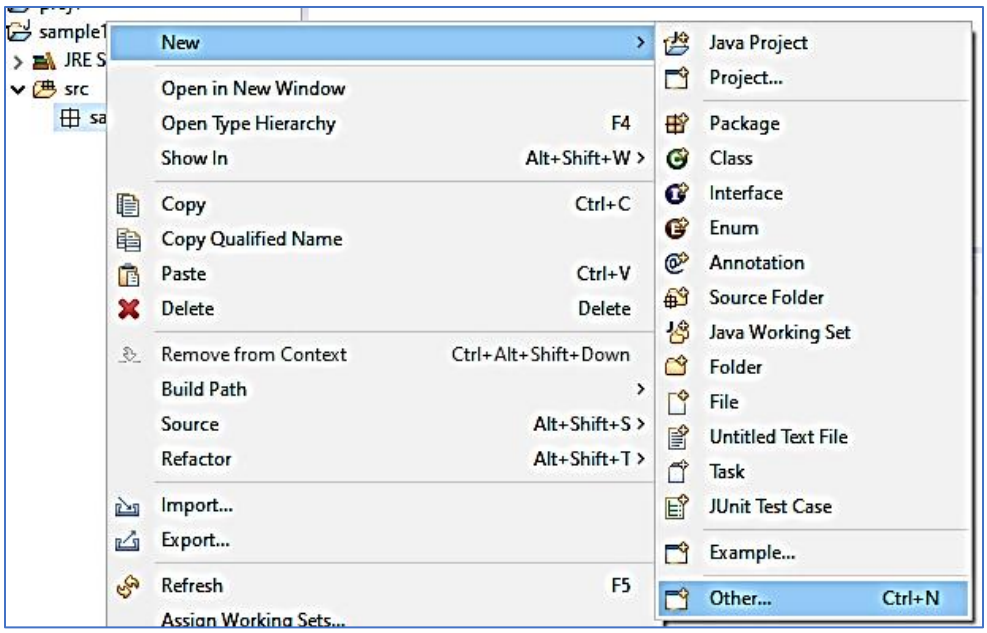
Custom Exception Usage

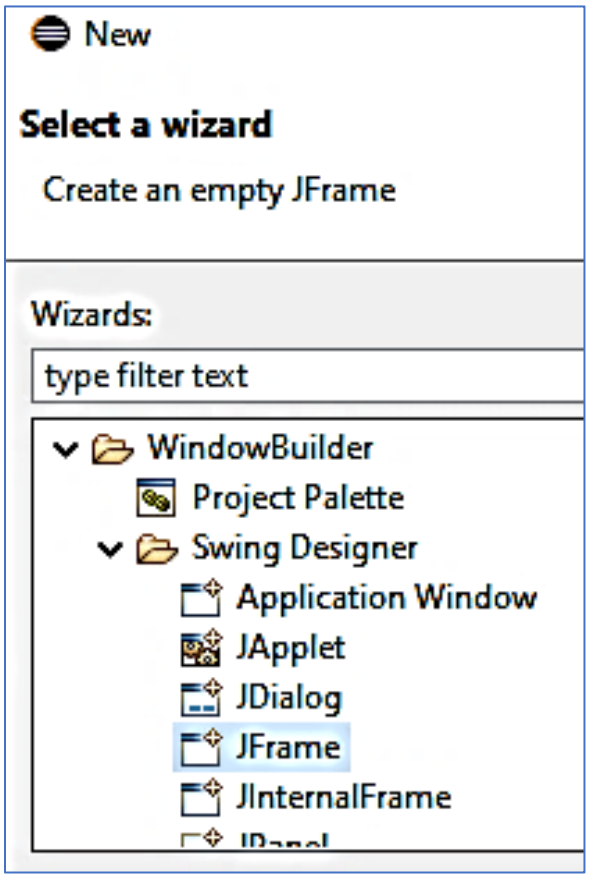
```
try {
   // Some condition
   throw new MyCustomException("Something went wrong");
} catch (MyCustomException e) {
   System.out.println(e.getMessage());
}
```

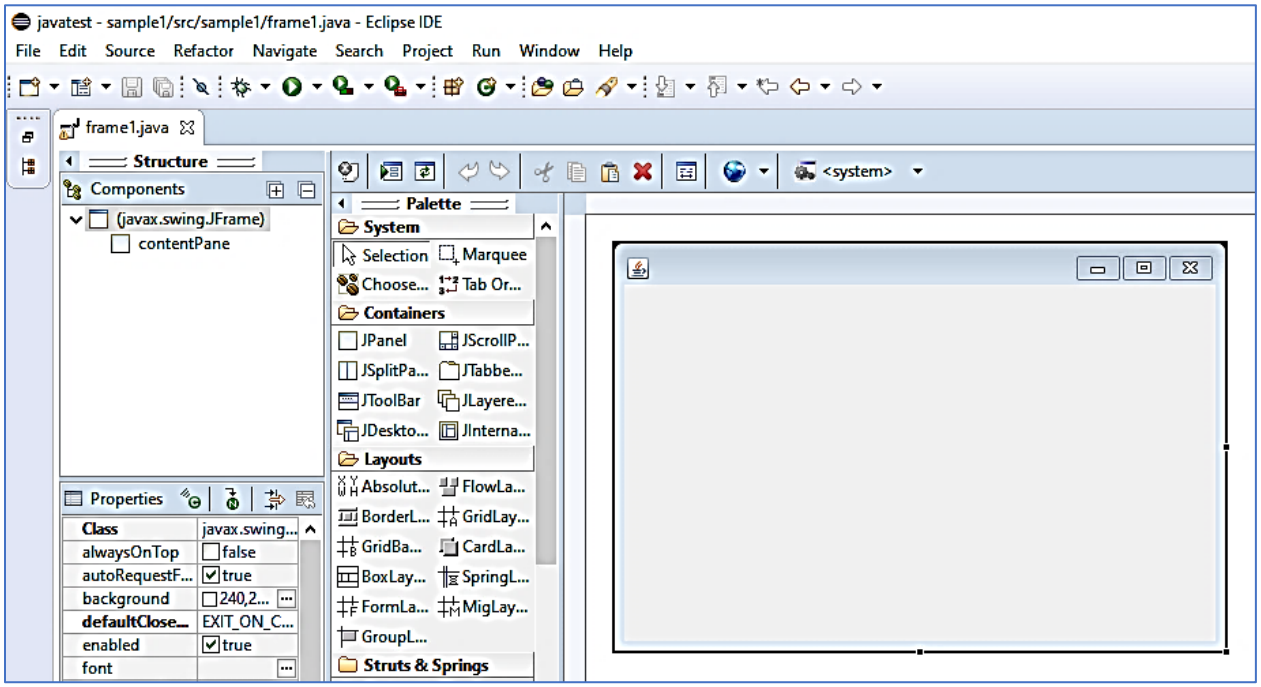
Setting up a JAVA GUI environment (eclipse-window builder)

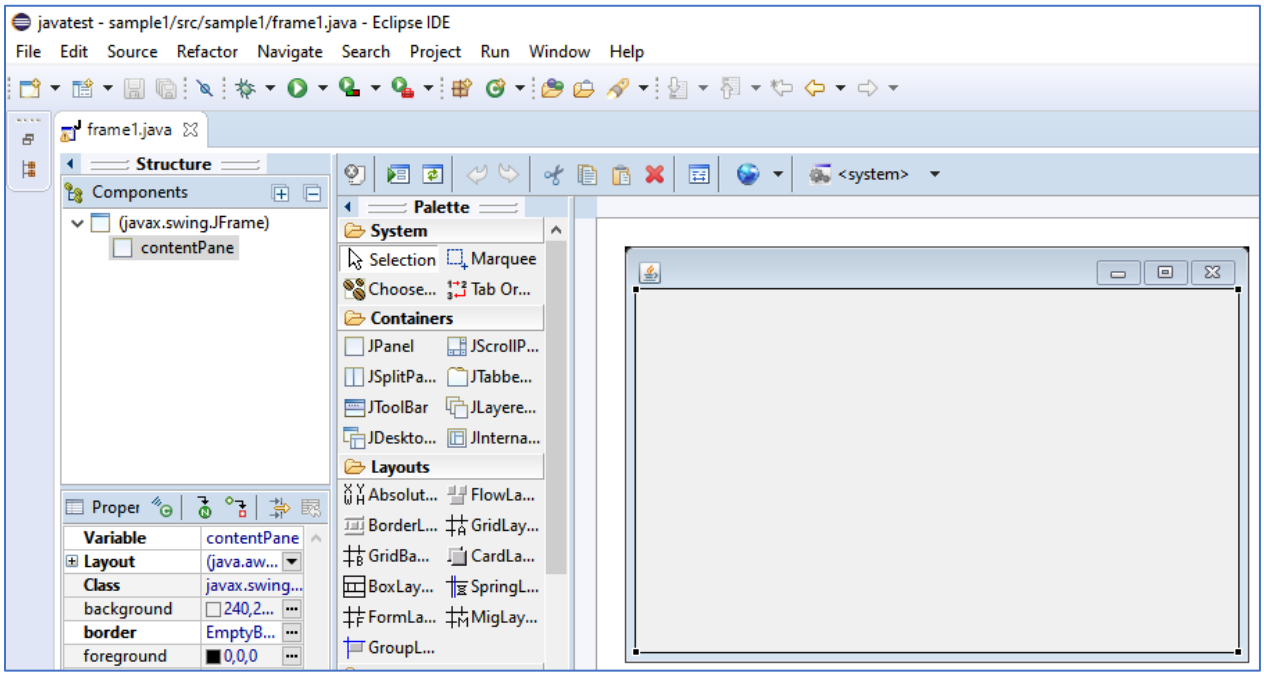
*note: For Eclipse(neon, oxygen and up) users, install windowbuilder from the eclipse marketplace

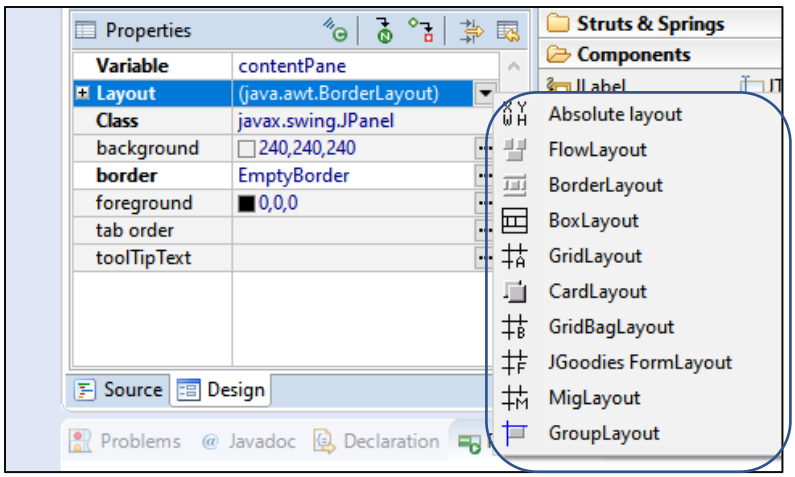



sample1
JRE S
src
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
Task
JUnit Test Case
Example...
Other... Ctrl+N


New
Select a wizard
Create an empty JFrame
Wizards:
type filter text
WindowBuilder
Project Palette
Swing Designer
Application Window
JApplet
JDialog
JFrame
JInternalFrame


javatest - sample1/src/sample1/frame1.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
frame1.java
Structure
Components
(javax.swing.JFrame)
contentPane
Palette
System
Selection
Marquee
Choose...
Tab Or...
Containers
JPanel
JScrollP...
JSplitPa...
JTabbe...
JToolBar
JLayere...
JDeskto...
JInterna...
Layouts
Absolut...
FlowLa...
BorderL...
GridLay...
GridBa...
CardLa...
BoxLay...
SpringL...
FormLa...
MigLay...
GroupL...
Struts & Springs
Properties
Class
javax.swing...
alwaysOnTop
false
autoRequestF...
true
background
240,2...
defaultClose...
EXIT_ON_C...
enabled
true
font
<system>

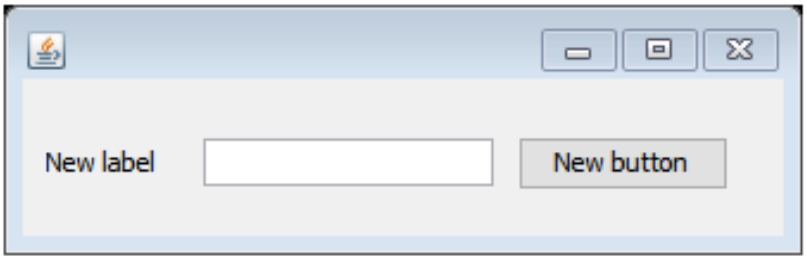

javatest - sample1/src/sample1/frame1.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
frame1.java
Structure
Components
(javax.swing.JFrame)
contentPane
Palette
System
Selection
Marquee
Choose...
Tab Or...
Containers
JPanel
JScrollP...
JSplitPa...
JTabbe...
JToolBar
JLayere...
JDeskto...
JInterna...
Layouts
Absolut...
FlowLa...
BorderL...
GridLay...
GridBa...
CardLa...
BoxLay...
SpringL...
FormLa...
MigLay...
GroupL...
Variable
contentPane
Layout
(java.aw...
Class
javax.swing...
background
240,2...
border
EmptyB...
foreground
0,0,0
<system>

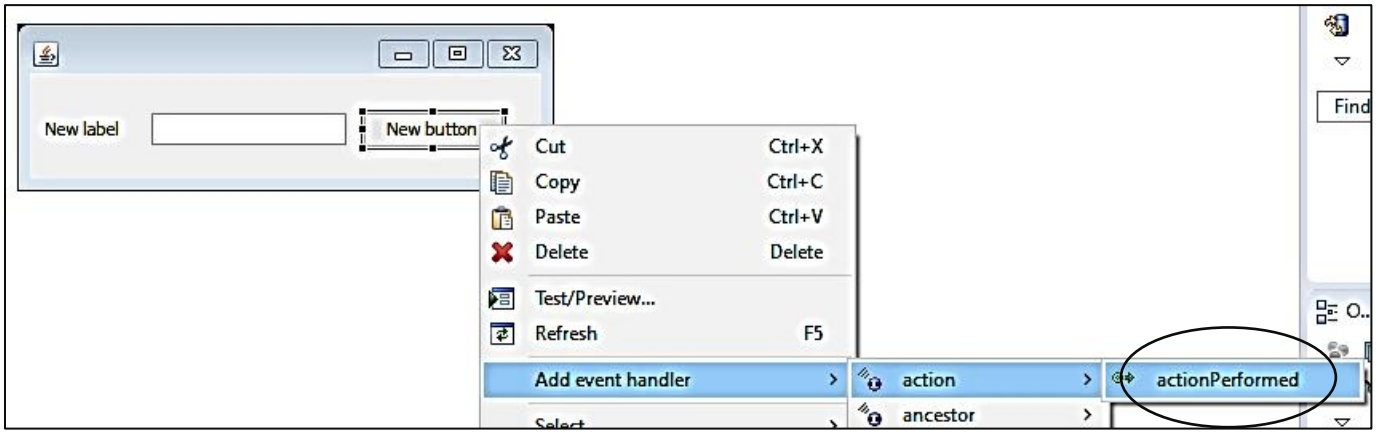



```
public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					frame1 frame = new frame1();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}

	/
	 * Create the frame.
	 */
	public frame1() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 450, 300);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		contentPane.setLayout(new BorderLayout(0, 0));
		setContentPane(contentPane);
	}
```

JLabel, JTextField, and JButton

*notable properties: setText(), getText(), setEnable(bool), setVisible(bool), setEditable(bool)



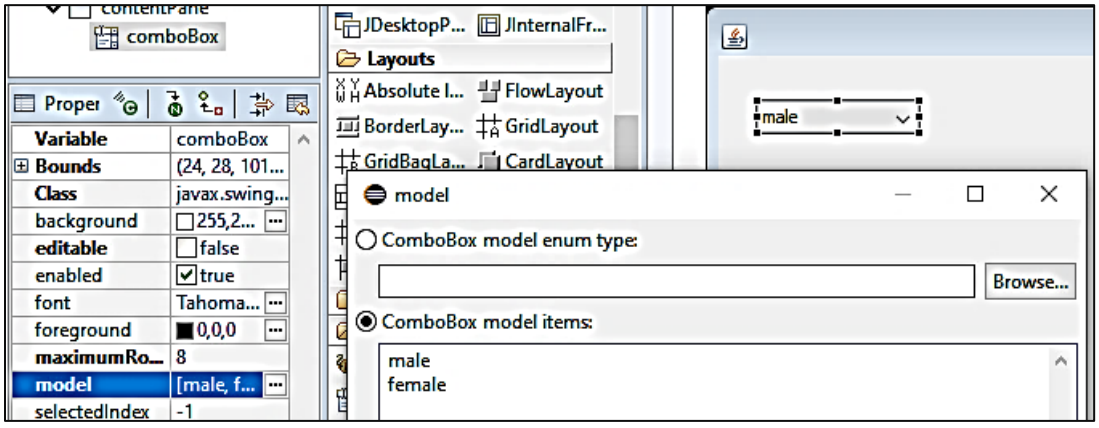



```
public class frame1 extends JFrame {

	private JPanel contentPane;
	private JTextField textField;

	/
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					frame1 frame = new frame1();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}

	/
```

```
	 * Create the frame.
	 */
	public frame1() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 336, 105);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(null);

		JLabel lblNewLabel = new JLabel("New label");
		lblNewLabel.setBounds(10, 28, 46, 14);
		contentPane.add(lblNewLabel);

		textField = new JTextField();
		textField.setBounds(76, 25, 122, 20);
		contentPane.add(textField);
		textField.setColumns(10);

		JButton btnNewButton = new JButton("New button");
		btnNewButton.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
			}
		});
		btnNewButton.setBounds(208, 24, 89, 23);
		contentPane.add(btnNewButton);
	}
}
```

JComboBox, JCheckBox, JRadioButton, JList

Editing JComboBox model:

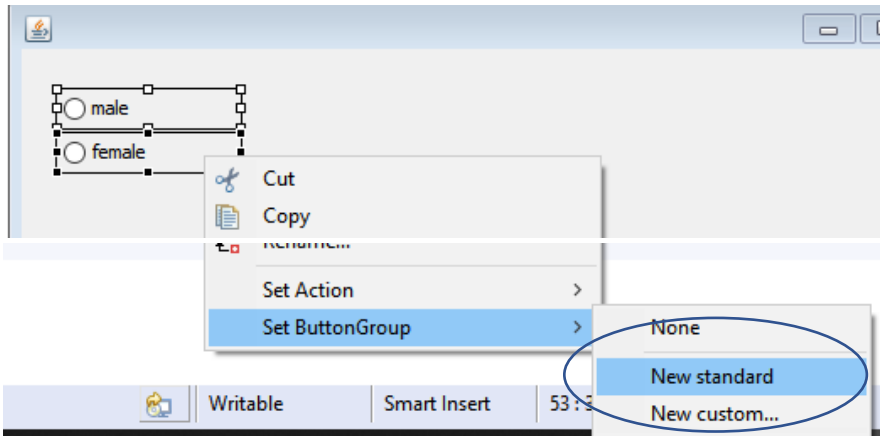



```
//to get the JComboBox selected item:
```

```
String choice = (String)combo.getSelectedItem();
```

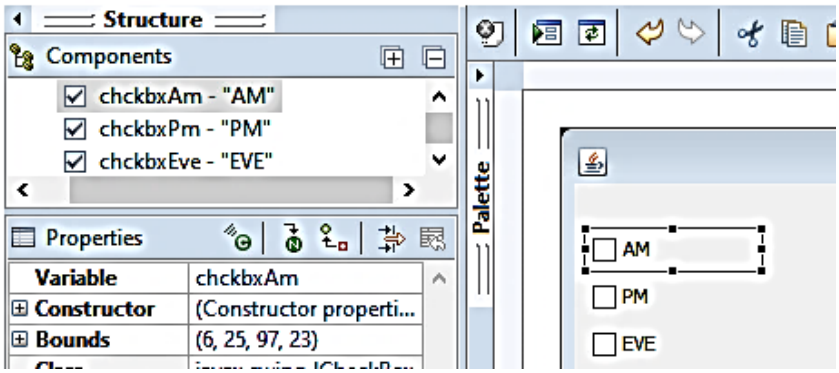
JRadioButton

//to toggle between JRadioButtons, place them in a Button group




```
//sample usage:
if(radio1.isSelected()){   //action;    }
```

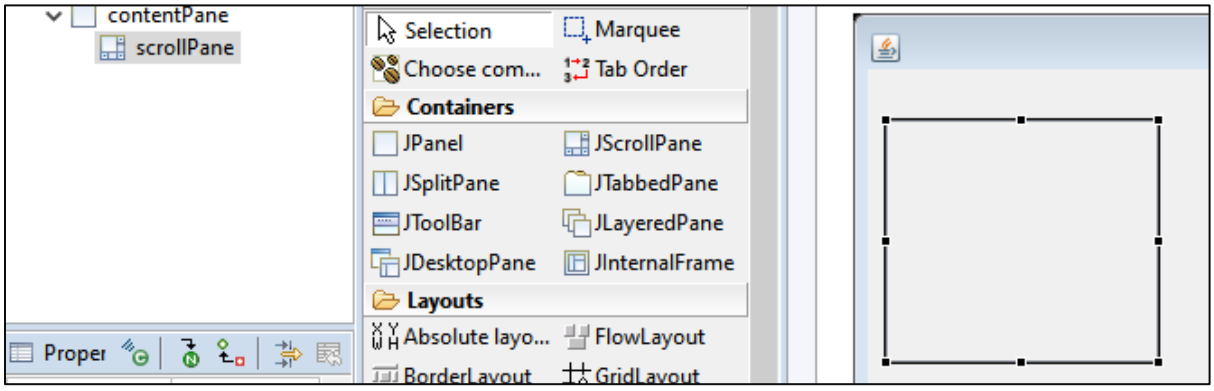
JCheckBox



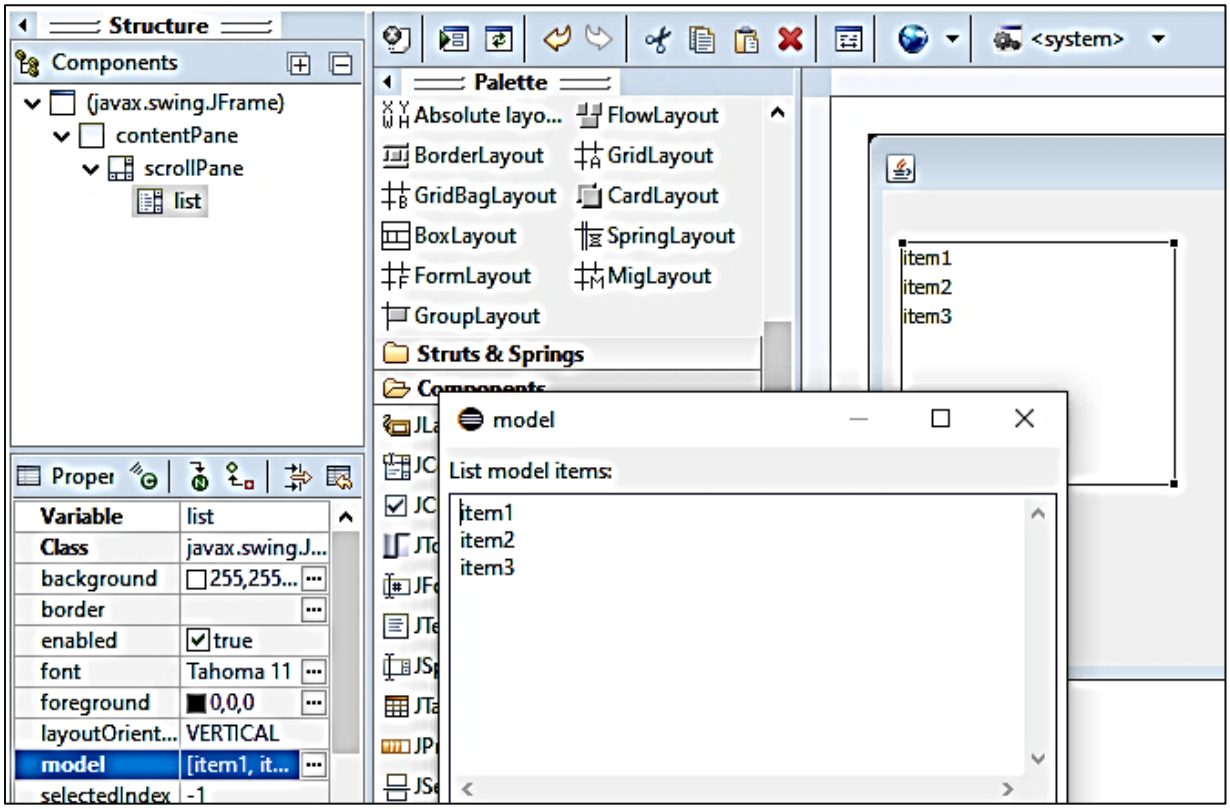

```
//sample usage:
if(check1.isSelected()){  //action;    }
```

JList

//add a JScrollPane first (which will contain the JList)



//add JList to the JScrollPane's viewport section then add item to JList using the model property




```
//to get the selected item from the JList
String choice = (String)list.getSelectedValue();
```

Using JOptionPane

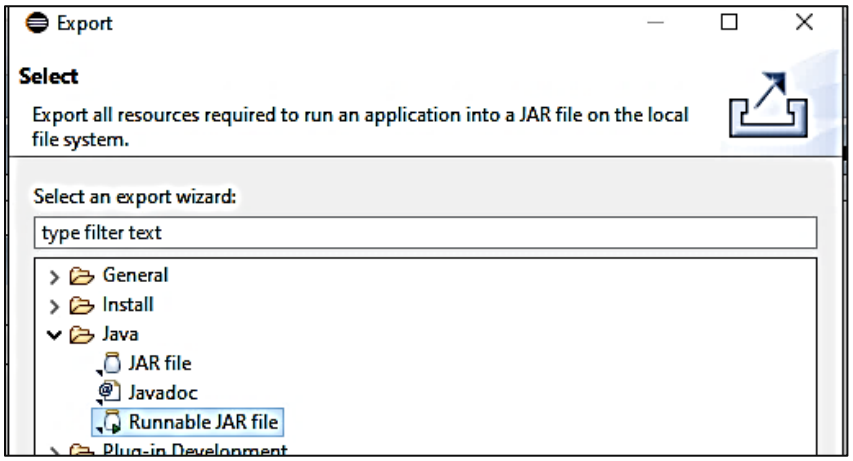
```
//message box
JOptionPane.showMessageDialog(<parent componet>/null,<message>,<title>,JOptionPane.<icon type>);
```

```
//input dialog
String input1 = JOptionPane.showInputDialog((<parent component>/null,<message>,<title>);
```

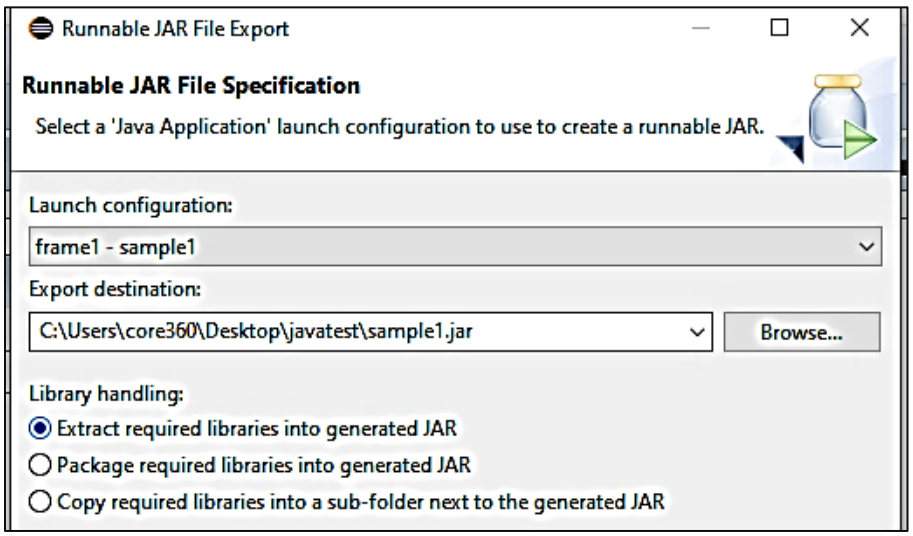
```
//confirm dialog
int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
if(a==JOptionPane.YES_OPTION){
   //action;
}
```

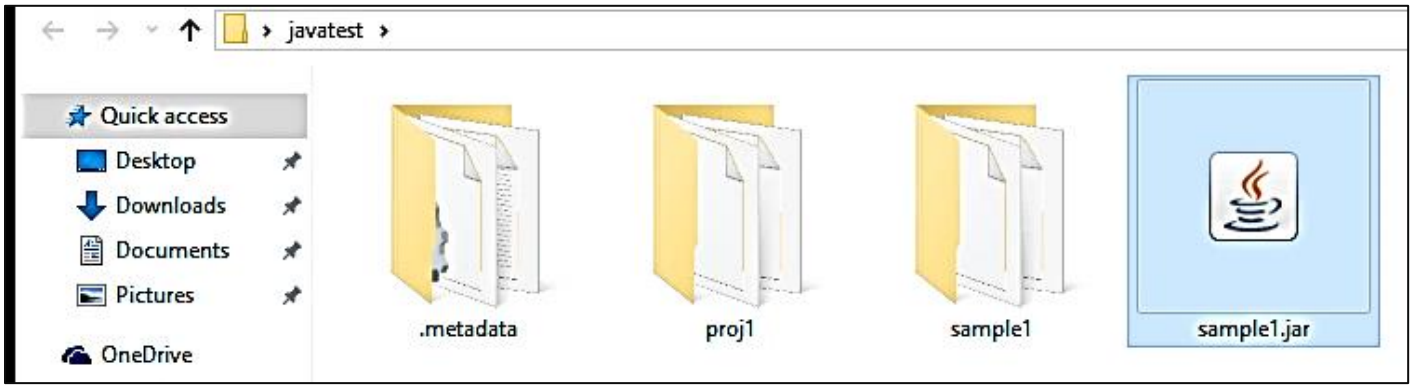
Exporting JAVA project as a runnable JAR program

1. Test run the app
2. File→export
3. Choose export type:



4. Choose launch configuration and set destination

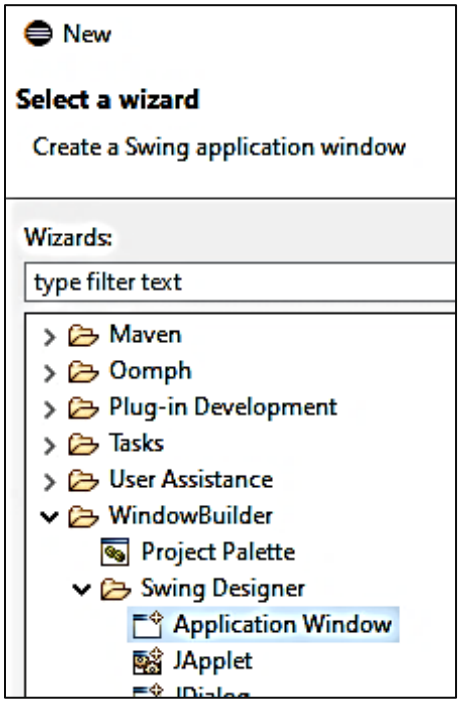




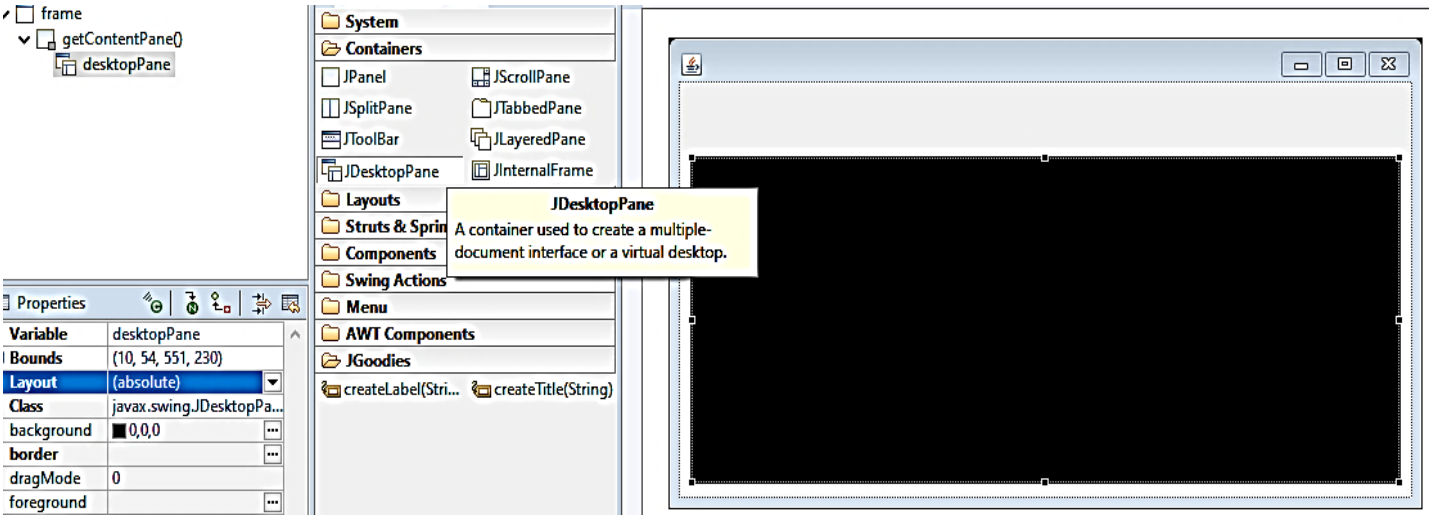
Creating a multi-document interface (MDI)

The application window (review of content pane, layout and other JFrame properties)

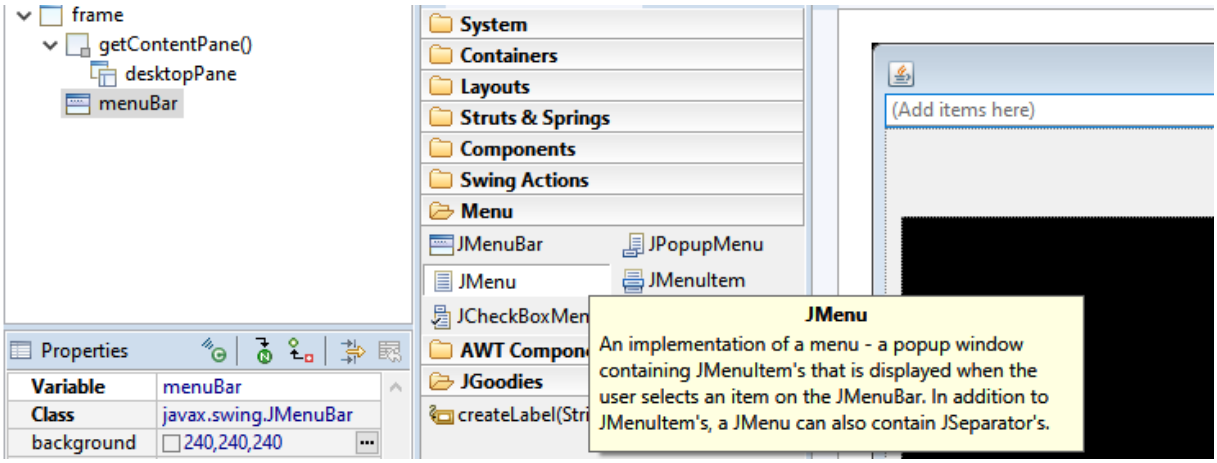
//create the parent frame, set layout of contentpane



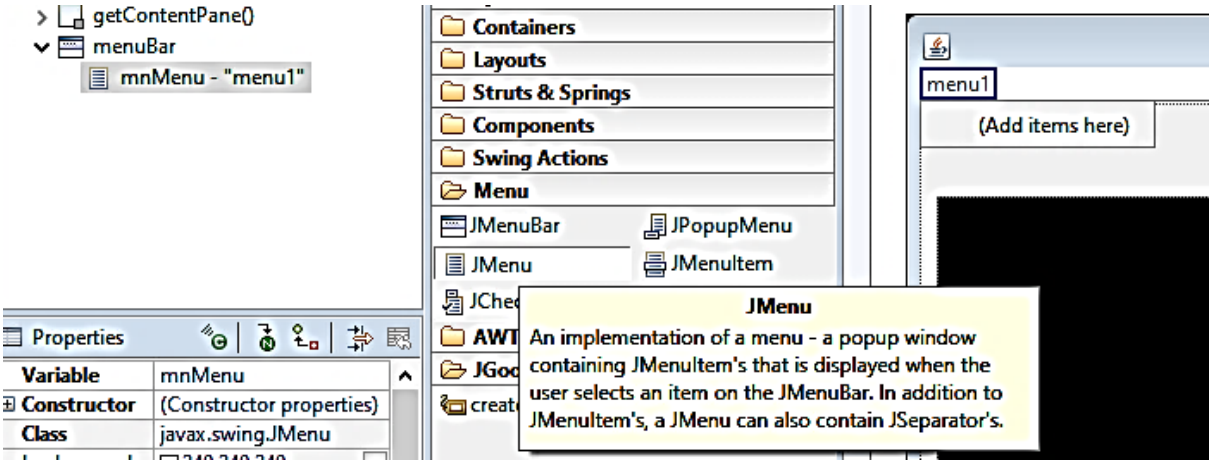
//add a JDesktopPane



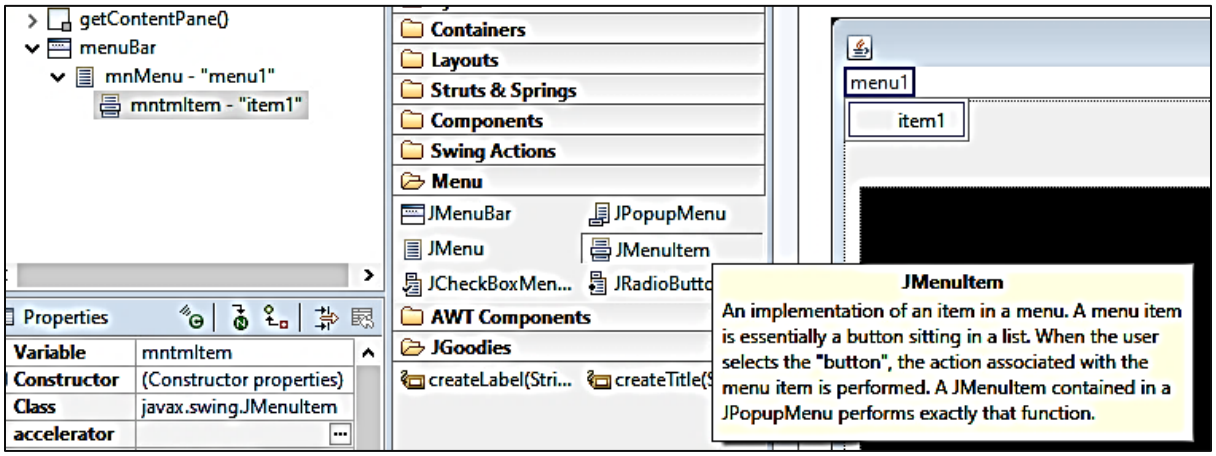
//add a JMenuBar



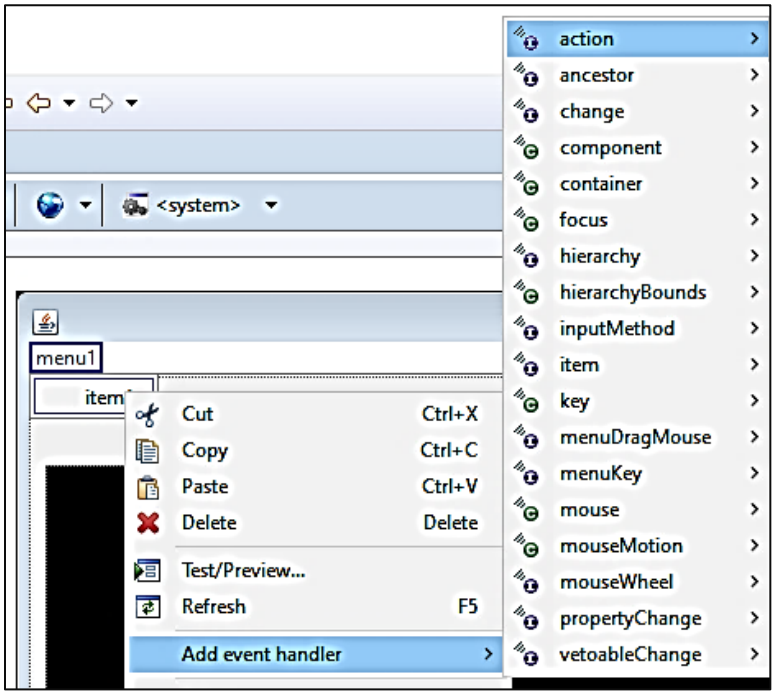
//add a JMenu unto the JMenuBar



//add JMenuItem to JMenu



//add event to JMenuItem



The JInternalFrame

1. rclick package→new→other(windowbuilder→JInternalFrame)
2. set JInternalFrame properties: closable,iconifiable,maximizable
3. back to application window class:

```
//move JDesktopPane declaration to class-level and add code to call the JInternalFrame class and show in JDesktopPane
public class app1 {

	…
	private JDesktopPane desktopPane;

	…
	public static void main(String[] args) {

	public app1() {
		initialize();
	}

	private void initialize() {
		frame = new JFrame();
		frame.setBounds(100, 100, 587, 334);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.getContentPane().setLayout(null);
```

```
            desktopPane = new JDesktopPane();
            desktopPane.setBounds(10, 54, 551, 230);
            frame.getContentPane().add(desktopPane);

        …
        JMenuItem mntmItem = new JMenuItem("item1");
        mntmItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                internal1 = new internal1();
                desktopPane.add(internal1);
                internal1.show();
                }
            });
            mnMenu.add(mntmItem);

        …
```

Java File Handling

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.*;
//import all the io namespace!

public class filehandling1 {

    public static void main(String[] args) {
        //create a file object
        //if you include other folders as part of the path, those folders should already be existing
        File text100 = new File("C:\\samplefile.txt");

        //create a file, surround it with try-catch
        try {
            text100.createNewFile();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
//read the file
//first, create an input file stream using FileReader
FileReader fr;
//then read the contents of the file using BufferedReadr
BufferedReader br;
//create a variable to hold the contents
String content;

try{
	//assign the file for the filereader
	fr = new FileReader("C:\\samplefile.txt");
	//assign the filereader object for the buffered reader
	br = new BufferedReader(fr);

	//loop to get the contents
	while((content = br.readLine()) != null)
	{
		System.out.println(content);
	}

	//dont forget to close the streams
	br.close();
	fr.close();

}
catch(Exception ex){}
```

```
//write to the file
//create another file stream using filewriter
FileWriter fw;
//then create a buffer writer
BufferedWriter bw;

String newstring = "im a new string";

//surround the process with try-catch
//assign to the filewriter the file to be edited
try{
fw = new FileWriter("C:\\samplefile.txt",true);//adding true uses the filewriter in append mode
bw = new BufferedWriter(fw);
bw.append(newstring + "\r\n");//using carriage "\n" is a tech for creating newline
```

```
                bw.close();
                fw.close();

                }

                catch(Exception ex){}
        }

}
```

```
//file copy, moving, deleting
Files. move("FileA", "FileB", StandardCopyOption. REPLACE_EXISTING) ;
Files. move("DirA", "DirB", StandardCopyOption. ATOMIC_MOVE) ;
Files. delete("path") ;
```

Managing and Reading a Properties file

Read a Properties File

Create a Properties File: This is a simple text file containing key-value pairs. For example, config.properties might look like this:

```
# This is a comment
database.url=jdbc:mysql://localhost:3306/mydb
database.user=root
database.password=pass123
```

Load Properties from the File: Use the Properties class to load key-value pairs.

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class ConfigReader {
   public static void main(String[] args) {
      Properties prop = new Properties();
      try {
         // Load the properties file
         prop.load(new FileInputStream("config.properties"));

         // Read properties
         String url = prop.getProperty("database.url");
         String user = prop.getProperty("database.user");
         String password = prop.getProperty("database.password");
```

```
        // Print the read properties
        System.out.println("URL: " + url);
        System.out.println("User: " + user);
        System.out.println("Password: " + password);
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```

Writing to a Properties File

You can also write to properties files using the Properties class. This is useful for setting configuration programmatically.

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Properties;

public class ConfigWriter {
  public static void main(String[] args) {
    Properties prop = new Properties();
    try {
      // Set properties
      prop.setProperty("database.url", "jdbc:mysql://localhost:3306/mydb");
      prop.setProperty("database.user", "root");
      prop.setProperty("database.password", "pass123");

      // Save properties to a file
      prop.store(new FileOutputStream("config.properties"), "Database Configuration");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Reading and writing XLS Files with POI API

Setup Apache POI

First, you need to include the Apache POI library in your project. If you're using Maven, add the following dependencies to your pom.xml file:

```
<!-- Apache POI for working with Excel files -->
<dependency>
  <groupId>org.apache.poi</groupId>
```

```
   <artifactId>poi</artifactId>
   <version>5.2.2</version> <!-- Check for the latest version -->
</dependency>
<dependency>
   <groupId>org.apache.poi</groupId>
   <artifactId>poi-ooxml</artifactId>
   <version>5.2.2</version> <!-- Check for the latest version -->
</dependency>
```

Reading an XLS File

To read from an .xls file, you use the HSSFWorkbook, HSSFSheet, and HSSFRow classes from Apache POI.

```
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.ss.usermodel.Cell;
import java.io.FileInputStream;
import java.io.IOException;

public class ReadXLSFile {
   public static void main(String[] args) {
      try (FileInputStream file = new FileInputStream("example.xls")) {
         HSSFWorkbook workbook = new HSSFWorkbook(file);
         HSSFSheet sheet = workbook.getSheetAt(0); // Get the first sheet

         // Iterate through each rows one by one
         sheet.forEach(row -> {
            row.forEach(cell -> {
               // Check the cell type and format accordingly
               switch (cell.getCellType()) {
                  case STRING:
                     System.out.print(cell.getStringCellValue() + " ");
                     break;
                  case NUMERIC:
                     System.out.print(cell.getNumericCellValue() + " ");
                     break;
                  case BOOLEAN:
                     System.out.print(cell.getBooleanCellValue() + " ");
                     break;
                  default:
                     System.out.print("Unknown Type ");
                     break;
               }
            });
```

```
            System.out.println();
        });
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```

Writing to an XLS File

To write to an .xls file, use HSSFWorkbook to create a workbook, and HSSFSheet to create a sheet in the workbook.

```
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.ss.usermodel.Cell;
import java.io.FileOutputStream;
import java.io.IOException;

public class WriteXLSFile {
  public static void main(String[] args) {
    HSSFWorkbook workbook = new HSSFWorkbook();
    HSSFSheet sheet = workbook.createSheet("Sample sheet");

    // Create a row and put some cells in it
    HSSFRow row = sheet.createRow(0);
    Cell cell = row.createCell(0);
    cell.setCellValue("Name");
    cell = row.createCell(1);
    cell.setCellValue("Age");

    // Write the workbook to an output stream
    try (FileOutputStream out = new FileOutputStream("new_example.xls")) {
      workbook.write(out);
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      try {
        workbook.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
}
```

Java JFileChooser Class

*you can use this control to browse your local directories and get absolute path of folders and files

```
final JFileChooser chooser = new JFileChooser();

//in a button action
int result=chooser.showOpenDialog(frmTest);//name of container JFrame of JApplet
if(result==JFileChooser.APPROVE_OPTION)
{
  File newfile = chooser.getSelectedFile();

  JOptionPane.showMessageDialog(null, newfile.getAbsolutePath());
}
else
{
  JOptionPane.showMessageDialog(null, "cancelled file open");
}
```

Java Collections API

The Java Collections Framework (JCF) is a set of classes and interfaces that implement commonly reusable collection data structures. It resides in the java.util package and is one of the most important and used aspects of the Java language. The Collections Framework provides both interfaces that define various collections and classes that implement them.

Core Interfaces

The Collections Framework is built around a set of core interfaces, each providing a different type of collections behavior:

- Collection Interface: The root of the collection hierarchy. It represents a group of objects known as its elements.
- List Interface: An ordered collection (also known as a sequence). The user can access elements by their integer index (position), and the list can contain duplicate elements.
- Set Interface: A collection that cannot contain duplicate elements. It models the mathematical set abstraction.
- SortedSet Interface: An extension of the Set interface that provides a total ordering on its elements.
- Map Interface: An object that maps keys to values. A map cannot contain duplicate keys, and each key can map to at most one value.
- SortedMap Interface: An extension of the Map interface that provides a total ordering of its keys.
- Core Classes

Some of the primary classes implementing these interfaces include:

- ArrayList and LinkedList: Implement the List interface. ArrayList provides a resizable-array, which can be thought of as a dynamic array. LinkedList implements a doubly-linked list.
- HashSet, LinkedHashSet, and TreeSet: Implement the Set interface. HashSet stores elements in a hash table, is the best-performing implementation, but makes no guarantees concerning the order of iteration. LinkedHashSet is a Hash table and linked list implementation that maintains insertion order. TreeSet provides a sorted set.

- HashMap, LinkedHashMap, and TreeMap: Implement the Map interface. HashMap offers a hash table-based implementation. LinkedHashMap maintains insertion order or access order. TreeMap provides a sorted map.

Using the Collections Framework

List Interface and ArrayList Class

The List interface allows you to create ordered collections, with ArrayList being a resizable-array implementation of the List interface.

```
import java.util.ArrayList;
import java.util.List;

public class ArrayListExample {
   public static void main(String[] args) {
      // Create an ArrayList
      List<String> list = new ArrayList<>();

      // Add elements
      list.add("Red");
      list.add("Green");
      list.add("Blue");

      // Access elements
      System.out.println("The element at index 1: " + list.get(1));

      // Modify elements
      list.set(1, "Yellow");

      // Remove elements
      list.remove("Blue");

      // Iterate over elements
      System.out.println("Updated list:");
      for (String color : list) {
         System.out.println(color);
      }
   }
}
```

Set Interface and HashSet Class

The Set interface represents a collection that cannot contain duplicate elements, and HashSet provides an implementation which uses a hash table for storage.

```
import java.util.HashSet;
import java.util.Set;

public class HashSetExample {
   public static void main(String[] args) {
      // Create a HashSet
      Set<Integer> set = new HashSet<>();

      // Add elements
      set.add(10);
      set.add(20);
      set.add(30);
      set.add(20); // This element will be ignored

      // Check if an element exists
      if (set.contains(30)) {
         System.out.println("30 is in the set.");
      }

      // Remove an element
      set.remove(10);

      // Iterate over elements
      System.out.println("Elements in set:");
      for (Integer number : set) {
         System.out.println(number);
      }
   }
}
```

Map Interface and HashMap Class

The Map interface represents a collection of key-value pairs, with HashMap being an implementation that uses a hash table for storage. It allows null values and one null key.

```
import java.util.HashMap;
import java.util.Map;

public class HashMapExample {
   public static void main(String[] args) {
      // Create a HashMap
      Map<String, Integer> map = new HashMap<>();

      // Add key-value pairs
      map.put("Alice", 30);
```

```
        map.put("Bob", 25);
        map.put("Charlie", 28);

        // Access a value
        int age = map.get("Alice");
        System.out.println("Alice's age: " + age);

        // Replace a value
        map.replace("Bob", 26);

        // Remove a key-value pair
        map.remove("Charlie");

        // Iterate over key-value pairs
        System.out.println("Updated map:");
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}
```

Java Reflections API

Java Reflection API allows programs to inspect and modify their own structure and behavior at runtime. This powerful feature can be used for a variety of purposes, such as inspecting classes, interfaces, fields, and methods; invoking methods at runtime; and creating new instances of classes. However, it's important to use reflection judiciously because it can lead to code that is hard to read and maintain, and it may have performance overhead.

Key Components of Reflection API

The Reflection API mainly revolves around the following classes and interfaces in the java.lang.reflect package:

- Class Class: Represents classes and interfaces in a running Java application.
- Field Class: Represents fields of a class or interface.
- Method Class: Represents methods of a class or interface.
- Constructor Class: Represents constructors of a class.

Basic Uses of Reflection

Accessing Class Information

You can obtain the Class object for a class using the getClass() method on an object instance, or by using ClassName.class. From this Class object, you can get a lot of information about the class.

```
Class<?> cls = String.class;
System.out.println("Class name: " + cls.getName());
System.out.println("Simple name: " + cls.getSimpleName());
System.out.println("Package name: " + cls.getPackage().getName());
```

Creating Instances Dynamically

You can create an instance of a class dynamically using its Class object.

```
Class<?> cls = Class.forName("java.lang.String");
Object obj = cls.getDeclaredConstructor(String.class).newInstance("Hello");
System.out.println(obj); // Outputs: Hello
```

Accessing Fields, Methods, and Constructors

Reflection API allows you to inspect and invoke methods, access fields, and instantiate objects using constructors.

```
// Accessing fields
Class<?> cls = MyClass.class;
Field field = cls.getField("myField");

// Accessing methods
Method method = cls.getMethod("myMethod", String.class);

// Accessing constructors
Constructor<?> constructor = cls.getConstructor();
```

Modifying Field Values

You can modify the value of a field in an object.

```
Class<?> cls = MyClass.class;
Field field = cls.getField("myField");
MyClass obj = new MyClass();
field.set(obj, newValue); // Sets the field's value for obj object
```

Invoking Methods

Methods can be invoked on objects.

```
Class<?> cls = MyClass.class;
Method method = cls.getDeclaredMethod("myMethod", parameterTypes);
MyClass obj = new MyClass();
Object returnValue = method.invoke(obj, methodArgs);
```

Important Considerations

- Performance: Reflection involves a significant overhead and should be used sparingly, especially in performance-sensitive areas of your application.
- Security: Using reflection can inadvertently expose sensitive parts of the code or data. Ensure appropriate security measures are in place.
- Access Control: Reflection can bypass normal access controls like private and protected modifiers, leading to potential security holes if not handled carefully.

Example: Inspecting Class Information

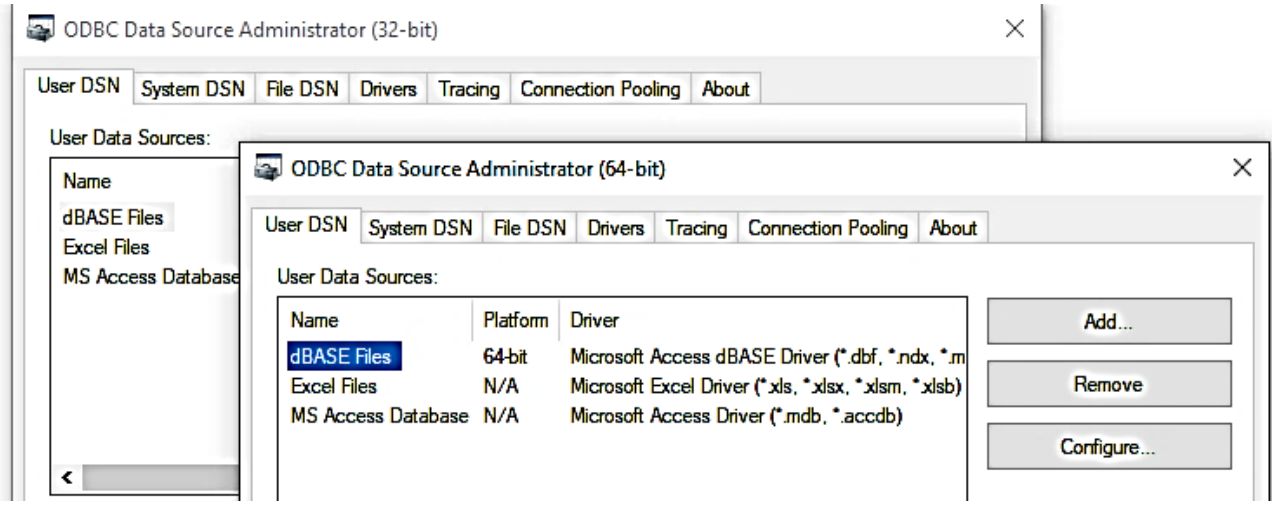
```
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.lang.reflect.Field;

public class ReflectionExample {
   public static void main(String[] args) throws ClassNotFoundException {
      Class<?> cls = Class.forName("java.util.ArrayList");

      System.out.println("Methods of " + cls.getName() + ":");
      for (Method method : cls.getDeclaredMethods()) {
         System.out.println(Modifier.toString(method.getModifiers()) + " " + method.getName());
      }

      System.out.println("\nFields of " + cls.getName() + ":");
      for (Field field : cls.getDeclaredFields()) {
         System.out.println(Modifier.toString(field.getModifiers()) + " " + field.getName());
      }
   }
}
```

Databases Concepts (MySQL, MSSQL and ORACLE DB)

Back-end Database Preparation (Database product specific)

- Create a database
- Setup tables, fields and indexes

Configuring data source name for TYPE – 1 JDBC Driver(DSN) for ODBC



Connecting to database from JAVA program

a. import library (java.sql.*;)

b. declare variables

```
Connection con;          //get connection to database
Statement st;    //variable for the sql statements
ResultSet rs;     //query result based on the sql statement
```

c. create constructor

```
public newclassname ()  //new object
{ connect(); }              //calling the connection method

public void connect()     //the connection method
{
try{
String driver = “sun.jdbc.odbc.JdbcOdbcDriver”;
Class.forName(driver);
String db = “jdbc:odbc:yourdatabase”;
con = DriverManager.getConnection(db);
st = con.createStatement(ResultSet.TypeScrollInsensitive,ResultSet.ConcurUpdateable);
String sql = “select* from [tablename]”;
Rs=st.executeQuery (sql);

//test using console outputting your database records
while (rs.next())
{
String field1 = rs.getString(“fieldname1”);
String field2 = rs.getString(“fieldname2”);
String field3 = rs.getString(“fieldname3”);
System.out.print(field1 + “ “ + field2 + “ “ + field3);
}
}catch (Exception ex)
        { //what happens if error; }
```

Connecting to MYSQL database from a JAVA Program

Requirements

a. Mysql (pref. from wamp)

b. Mysql jdbc driver https://www.mysql.com/products/connector/

Creating a database

c. Set up xamp

d. Create tables,fields and indexes

Connecting to database from JAVA program

d. Using the mysql jdbc driver

(on JAVA project, r.click library build path conf. build path libraries add ext. jar browse for mysql connector jar)

e. declare variables

```
Connection con;          //get connection to database
Statement st;    //variable for the sql statements
ResultSet rs;      //query result based on the sql statement
```

f. create constructor

```
public newclassname () //new object
{
  connect();                          //calling the connection method
}

public void connect()     //the connection method
{
try{

Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/yourdatabase","root","root");
PreparedStatement prepst = con.preparedStatement("select * from [tablename]);
ResultSet rs = st.executeQuery();

//test using console outputting your database records
while (rs.next())
{
String field1 = rs.getString("fieldname1");
String field2 = rs.getString("fieldname2");
String field3 = rs.getString("fieldname3");
System.out.print(field1 + " " + field2 + " " + field3);
}
}catch (Exception ex)
        { //what happens if error; }
```

Performing CRUD operations to database from JAVA program

```
//sample project
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
```

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import javax.swing.JTabbedPane;
import javax.swing.JPanel;
import java.awt.Color;
import javax.swing.JPasswordField;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.border.TitledBorder;
import javax.swing.border.LineBorder;

public class accessform1 {

	private JFrame frmSampleDatabaseApplication;
	private JButton btnSaveEdited;
	private JButton btnOpenEditing;
	private JButton btnFirst;
	private JButton btnPrevious;
	private JButton btnNext;
	private JButton btnLast;
	private JButton btnAdd;
	private JButton btnSave;
	private JButton btnDelete;

	//initial connection
	Connection con;
	Statement st;
	ResultSet rs;

	//for login table
	Statement loginst;
	ResultSet loginrs;

	private JTextField textField;
	private JTextField textField_1;
	private JTextField textField_2;
	private JTextField textField_3;
	private JTextField usernamefield;
	private JPasswordField passwordfld;
	private final ButtonGroup buttonGroup = new ButtonGroup();
	private JTable table;
```

```
/
 * Launch the application.
 */
public static void main(String[] args) {
	EventQueue.invokeLater(new Runnable() {
		public void run() {
			try {
				accessform1 window = new accessform1();
				window.frmSampleDatabaseApplication.setVisible(true);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	});
}

/
 * Create the application.
 */
public accessform1() {
	initialize();
	connect();
	initialvalues();
}


public void connect(){
	System.out.println ("you called the connect()");
	try{
		String driver= "sun.jdbc.odbc.JdbcOdbcDriver";
		Class.forName(driver);
		String db="jdbc:odbc:eclipsedatabase";
		con= DriverManager.getConnection(db);
		st=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);
		String sql = "select * from employeetable";
		rs=st.executeQuery(sql);
	}catch (Exception ex) {}
}

public void loginconnect(){

	System.out.println ("you called the loginconnect()");
	try{
		String driver= "sun.jdbc.odbc.JdbcOdbcDriver";
		Class.forName(driver);
		String db="jdbc:odbc:eclipsedatabase";
		con= DriverManager.getConnection(db);
		st=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);

	}catch (Exception ex) {}
```

```
}
public void initialvalues(){
	try {
		rs.next();
		textField.setText(rs.getString("fname"));
		textField_1.setText(rs.getString("mname"));
		textField_2.setText(rs.getString("lname"));
		textField_3.setText(rs.getString("position"));
	}catch (Exception ex){}
}



/
 * Initialize the contents of the frame.
 */
private void initialize() {

	frmSampleDatabaseApplication = new JFrame();
	frmSampleDatabaseApplication.setTitle("Sample Database Application");
	frmSampleDatabaseApplication.setBounds(100, 100, 584, 546);
	frmSampleDatabaseApplication.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	frmSampleDatabaseApplication.getContentPane().setLayout(null);

	final JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);
	tabbedPane.setBounds(10, 224, 548, 273);
	frmSampleDatabaseApplication.getContentPane().add(tabbedPane);
	tabbedPane.setVisible(true);

	JPanel panel = new JPanel();
	panel.setBackground(Color.WHITE);
	tabbedPane.addTab("Form", null, panel, null);
	panel.setLayout(null);

	JLabel label = new JLabel("first name: ");
	label.setBounds(47, 42, 55, 14);
	panel.add(label);

	textField = new JTextField();
	textField.setBounds(107, 39, 180, 20);
	textField.setText((String) null);
	textField.setEditable(false);
	textField.setColumns(10);
	panel.add(textField);

	JLabel label_1 = new JLabel("mid name: ");
	label_1.setBounds(47, 73, 52, 14);
	panel.add(label_1);
```

```
textField_1 = new JTextField();
textField_1.setBounds(107, 70, 180, 20);
textField_1.setText((String) null);
textField_1.setEditable(false);
textField_1.setColumns(10);
panel.add(textField_1);

JLabel label_2 = new JLabel("last name: ");
label_2.setBounds(46, 108, 53, 14);
panel.add(label_2);

textField_2 = new JTextField();
textField_2.setBounds(107, 105, 180, 20);
textField_2.setText((String) null);
textField_2.setEditable(false);
textField_2.setColumns(10);
panel.add(textField_2);

JLabel label_3 = new JLabel("position:");
label_3.setBounds(47, 142, 41, 14);
panel.add(label_3);

textField_3 = new JTextField();
textField_3.setBounds(107, 139, 180, 20);
textField_3.setText((String) null);
textField_3.setEditable(false);
textField_3.setColumns(10);
panel.add(textField_3);

btnFirst = new JButton("first");
btnFirst.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {

	try{

		rs.first();
		textField.setText(rs.getString("fname"));
		textField_1.setText(rs.getString("mname"));
		textField_2.setText(rs.getString("lname"));
		textField_3.setText(rs.getString("position"));

	}catch(Exception ex){}

	btnFirst.setEnabled(false);
	btnPrevious.setEnabled(false);
	btnNext.setEnabled(true);
	btnLast.setEnabled(true);

	}
```

```
});
btnFirst.setBounds(24, 191, 78, 23);
btnFirst.setEnabled(false);
panel.add(btnFirst);

btnPrevious = new JButton("previous");
btnPrevious.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {

		try{

			if (rs.previous())
			{
			textField.setText(rs.getString("fname"));
			textField_1.setText(rs.getString("mname"));
			textField_2.setText(rs.getString("lname"));
			textField_3.setText(rs.getString("position"));
			}
			else
			{
				JOptionPane.showMessageDialog(null, "first record");
				btnFirst.setEnabled(false);
				btnPrevious.setEnabled(false);
			}


		}catch(Exception ex){}


		btnNext.setEnabled(true);
		btnLast.setEnabled(true);

	}
});
btnPrevious.setBounds(100, 191, 73, 23);
btnPrevious.setEnabled(false);
panel.add(btnPrevious);

btnNext = new JButton("next");
btnNext.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
		try{

			if(rs.next())
			{
			textField.setText(rs.getString("fname"));
			textField_1.setText(rs.getString("mname"));
			textField_2.setText(rs.getString("lname"));
			textField_3.setText(rs.getString("position"));
			}
```

```
                        else
                        {
                                JOptionPane.showMessageDialog(null, "last record");
                                btnNext.setEnabled(false);
                                btnLast.setEnabled(false);
                        }

                }catch(Exception ex){}
                btnFirst.setEnabled(true);
                btnPrevious.setEnabled(true);

        }
});
btnNext.setBounds(172, 191, 78, 23);
panel.add(btnNext);

btnLast = new JButton("last");
btnLast.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {

                try{

                        rs.last();
                        textField.setText(rs.getString("fname"));
                        textField_1.setText(rs.getString("mname"));
                        textField_2.setText(rs.getString("lname"));
                        textField_3.setText(rs.getString("position"));

                }catch(Exception ex){}

                btnFirst.setEnabled(true);
                btnPrevious.setEnabled(true);
                btnNext.setEnabled(false);
                btnLast.setEnabled(false);
        }
});
btnLast.setBounds(249, 191, 72, 23);
panel.add(btnLast);

btnAdd = new JButton("add");
btnAdd.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {

        btnSave.setEnabled(true);
        btnAdd.setEnabled(false);
        btnOpenEditing.setEnabled(false);
        btnSaveEdited.setEnabled(false);
        btnDelete.setEnabled(false);
        btnFirst.setEnabled(false);
        btnPrevious.setEnabled(false);
```

```
		btnNext.setEnabled(false);
		btnLast.setEnabled(false);

		textField.setText("");
		textField_1.setText("");
		textField_2.setText("");
		textField_3.setText("");
		textField.setEnabled(true);
		textField_1.setEnabled(true);
		textField_2.setEnabled(true);
		textField_3.setEnabled(true);
		textField.setEditable(true);
		textField_1.setEditable(true);
		textField_2.setEditable(true);
		textField_3.setEditable(true);

		}
});
btnAdd.setBounds(419, 70, 89, 23);
panel.add(btnAdd);

btnSave = new JButton("save");
btnSave.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent arg0) {

			String newfirstname = textField.getText();
			String newmiddlename = textField_1.getText();
			String newlastname = textField_2.getText();
			String newposition = textField_3.getText();

			try{
				rs.last();
				rs.moveToInsertRow();
				rs.updateString("fname", newfirstname);
				rs.updateString("mname", newmiddlename);
				rs.updateString("lname", newlastname);
				rs.updateString("position", newposition);
				rs.insertRow();


				con.commit();
				rs.close();
				st.close();
				con.close();
				connect();


				rs.refreshRow();
				textField.setText(rs.getString("fname"));
				textField_1.setText(rs.getString("mname"));
```

```
                    textField_2.setText(rs.getString("lname"));
                    textField_3.setText(rs.getString("position"));


                }catch(Exception ex){}

                JOptionPane.showMessageDialog(null, "record update successful!");
                btnSave.setEnabled(false);
                btnAdd.setEnabled(true);
                btnOpenEditing.setEnabled(true);
                btnSaveEdited.setEnabled(false);
                btnDelete.setEnabled(true);
                btnFirst.setEnabled(true);
                btnPrevious.setEnabled(true);
                btnNext.setEnabled(true);
                btnLast.setEnabled(true);

        }
});
btnSave.setBounds(419, 104, 89, 23);
btnSave.setEnabled(false);
panel.add(btnSave);

btnOpenEditing = new JButton("open editing");
btnOpenEditing.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {

        textField.setEnabled(true);
        textField_1.setEnabled(true);
        textField_2.setEnabled(true);
        textField_3.setEnabled(true);
        btnOpenEditing.setVisible(false);
        btnSaveEdited.setVisible(true);
        btnSaveEdited.setEnabled(true);

        btnSave.setEnabled(false);
        btnAdd.setEnabled(false);
        btnDelete.setEnabled(false);
        btnFirst.setEnabled(false);
        btnPrevious.setEnabled(false);
        btnNext.setEnabled(false);
        btnLast.setEnabled(false);

        textField.setEditable(true);
        textField_1.setEditable(true);
        textField_2.setEditable(true);
        textField_3.setEditable(true);
        }
});
btnOpenEditing.setBounds(417, 138, 91, 23);
```

```
		panel.add(btnOpenEditing);

		btnDelete = new JButton("delete");
		btnDelete.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {


			int delconfirm = JOptionPane.showConfirmDialog(null, "Are you sure you want to delete tis
record?", "Delete confirmation",JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
			if (delconfirm==JOptionPane.YES_OPTION)
			{
					try{
							rs.deleteRow();
							rs.refreshRow();


								textField.setText(rs.getString("fname"));
								textField_1.setText(rs.getString("mname"));
								textField_2.setText(rs.getString("lname"));
								textField_3.setText(rs.getString("position"));


					}catch(Exception ex){}

			}
			else
			{
				JOptionPane.showMessageDialog(null, "whew, that was close","good thing i
asked!",JOptionPane.INFORMATION_MESSAGE);
			}

			}
		});
		btnDelete.setBounds(419, 173, 89, 23);
		panel.add(btnDelete);

		btnSaveEdited = new JButton("save edited");
		btnSaveEdited.setBounds(419, 140, 89, 23);
		panel.add(btnSaveEdited);
		btnSaveEdited.setVisible(false);
		btnSaveEdited.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {


				//get current displayed values and set to new variables
				String firstname = textField.getText();
				String middlename = textField_1.getText();
				String lastname = textField_2.getText();
				String position = textField_3.getText();
```

```
                        try{

                                rs.updateString("fname", firstname);
                                rs.updateString("mname", middlename);
                                rs.updateString("lname", lastname);
                                rs.updateString("position", position);
                                rs.updateRow();
                                rs.refreshRow();


                        }catch(Exception ex){}

                        JOptionPane.showMessageDialog(null, "record update successful!");
                        btnOpenEditing.setVisible(true);
                        btnSaveEdited.setVisible(false);
                        btnSave.setEnabled(false);
                        btnAdd.setEnabled(true);
                        btnDelete.setEnabled(true);
                        btnFirst.setEnabled(true);
                        btnPrevious.setEnabled(true);
                        btnNext.setEnabled(true);
                        btnLast.setEnabled(true);
                }
        });

        JPanel panel_1 = new JPanel();
        panel_1.setBackground(Color.WHITE);
        tabbedPane.addTab("Table", null, panel_1, null);
        panel_1.setLayout(null);

        table = new JTable();
        table.setBounds(21, 26, 448, 194);
        panel_1.add(table);


        JPanel panel_2 = new JPanel();
        panel_2.setBackground(Color.WHITE);
        tabbedPane.addTab("Report", null, panel_2, null);

        JPanel panel_3 = new JPanel();
        panel_3.setBackground(Color.WHITE);
        tabbedPane.addTab("search", null, panel_3, null);
        panel_3.setLayout(null);

        JPanel panel_4 = new JPanel();
        panel_4.setBackground(Color.WHITE);
        panel_4.setBorder(new TitledBorder(new LineBorder(new Color(0, 0, 0)), "search by",
TitledBorder.LEADING, TitledBorder.TOP, null, null));
        panel_4.setBounds(13, 28, 121, 153);
        panel_3.add(panel_4);
```

```
		panel_4.setLayout(null);

		JRadioButton rdbtnFirstName = new JRadioButton("first name");
		rdbtnFirstName.setBackground(Color.WHITE);
		rdbtnFirstName.setBounds(6, 16, 109, 23);
		panel_4.add(rdbtnFirstName);
		buttonGroup.add(rdbtnFirstName);

		JRadioButton rdbtnLastname = new JRadioButton("lastname");
		rdbtnLastname.setBackground(Color.WHITE);
		rdbtnLastname.setBounds(6, 42, 109, 23);
		panel_4.add(rdbtnLastname);
		buttonGroup.add(rdbtnLastname);

		JRadioButton rdbtnPosition = new JRadioButton("position");
		rdbtnPosition.setBackground(Color.WHITE);
		rdbtnPosition.setBounds(6, 68, 109, 23);
		panel_4.add(rdbtnPosition);
		buttonGroup.add(rdbtnPosition);

		JButton btnGo = new JButton("go!");
		btnGo.setBounds(26, 98, 66, 23);
		panel_4.add(btnGo);

		JLabel lblUsername = new JLabel("username");
		lblUsername.setBounds(62, 51, 65, 14);
		frmSampleDatabaseApplication.getContentPane().add(lblUsername);

		JLabel lblPassword = new JLabel("password");
		lblPassword.setBounds(62, 76, 65, 14);
		frmSampleDatabaseApplication.getContentPane().add(lblPassword);

		usernamefield = new JTextField();
		usernamefield.setBounds(133, 48, 102, 17);
		frmSampleDatabaseApplication.getContentPane().add(usernamefield);
		usernamefield.setColumns(10);

		passwordfld = new JPasswordField();
		passwordfld.setBounds(132, 73, 103, 17);
		frmSampleDatabaseApplication.getContentPane().add(passwordfld);

		JButton btnLogIn = new JButton("log in");
		btnLogIn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {

				try{

					loginconnect();
					String loginsql = "select * from login where username='" +
usernamefield.getText() + "' and password='" + passwordfld.getText() + "'";
```

```
                                        loginrs=st.executeQuery(loginsql);
                                        if (loginrs.next())
                                        {
                                                JOptionPane.showMessageDialog(null, "welcome");
                                                tabbedPane.setVisible(true);
                                        }
                                        else
                                        {
                                                JOptionPane.showMessageDialog(null, "sorry try again");
                                                usernamefield.setText(null);
                                                passwordfld.setText(null);
                                        }
                                }catch(Exception ex){}
                        }
                });
                btnLogIn.setBounds(133, 121, 89, 23);
                frmSampleDatabaseApplication.getContentPane().add(btnLogIn);
        }
}
```

Displaying records in a JTable

```
//once database connection has been established
public void Refresh_Data(){
                try {
                        String sqlQuery2="Select * from table100 order by name";
                        rs=st.executeQuery(sqlQuery2);

                        int i=4;
                        DefaultTableModel d=new DefaultTableModel();
                        d.addColumn("Name");
                        d.addColumn("Department");
                        d.addColumn("Position");
                        d.addColumn("Year Hired");
                        table.setModel(d);
                        String []s=new String[i];
                        while(rs.next()){
                                        s[0]=rs.getString("name");
                                        s[1]=rs.getString("department");
                                        s[2]=rs.getString("position");
                                        s[3]=rs.getString("yearhired");
                                        d.addRow(s);
                        }
                } catch(Exception e1){JOptionPane.showMessageDialog(null, e1);}
        }
```

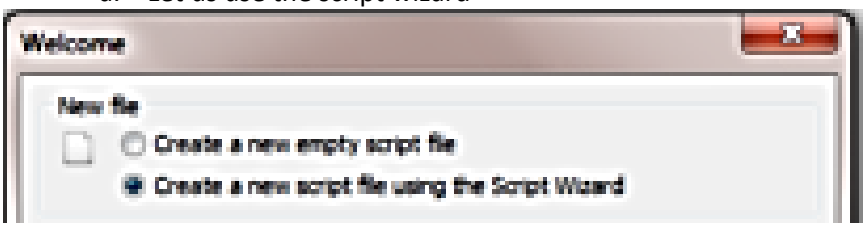
Running shell commands

```
//using the Runtime class
try
{
        Runtime.getRuntime().exec("notepad.exe")
}catch(IOException) {}
```

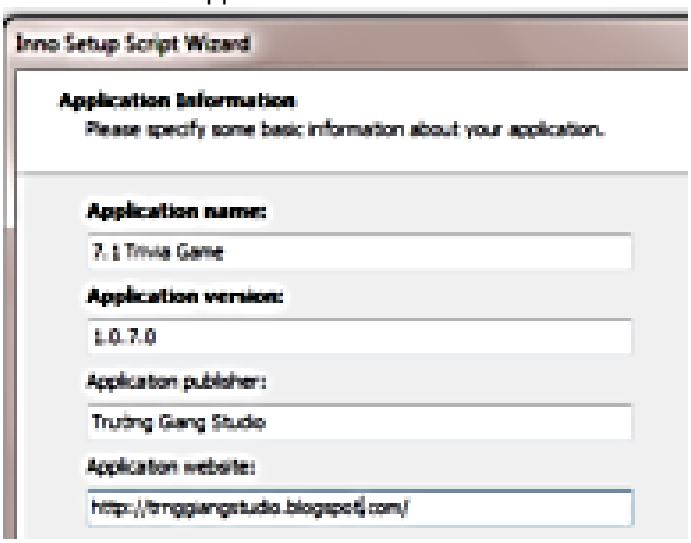
Creating a packager / installer for your java application

1. Download innosetup (http://www.jrsoftware.org/isdl.php) and install
2. Test and export a runnable jar from your java application
3. Launch innosetup compiler and follow the wizard

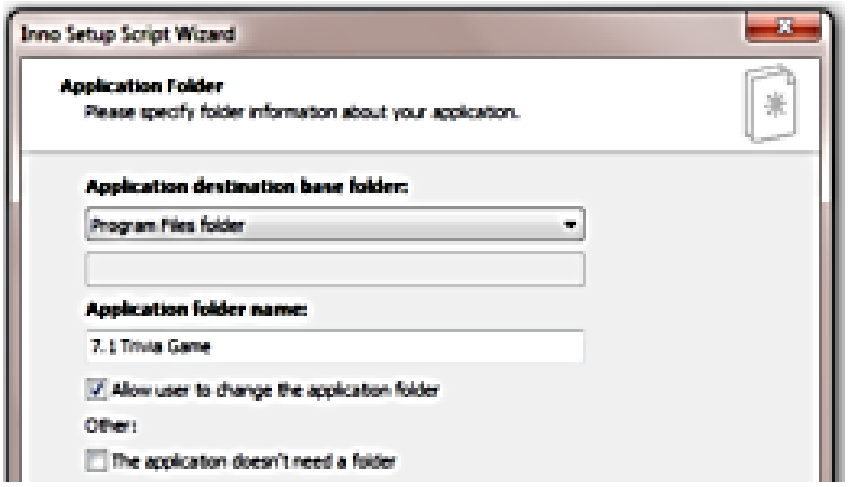
    a. Let us use the script wizard



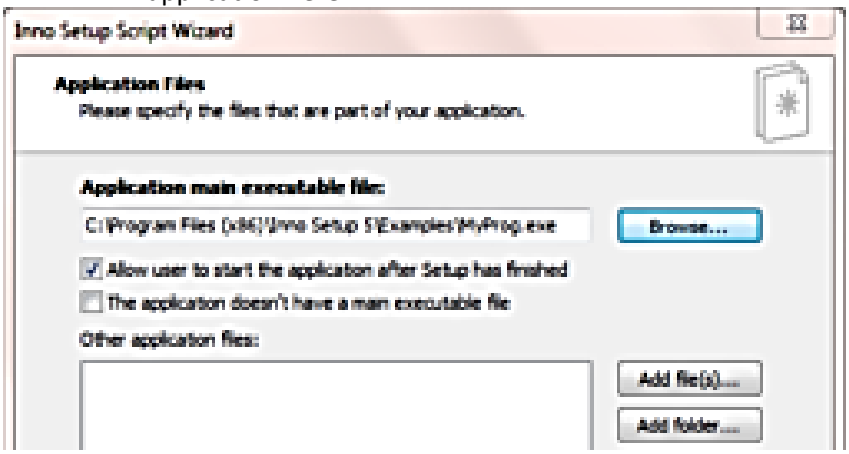
    b. Set application name and other details



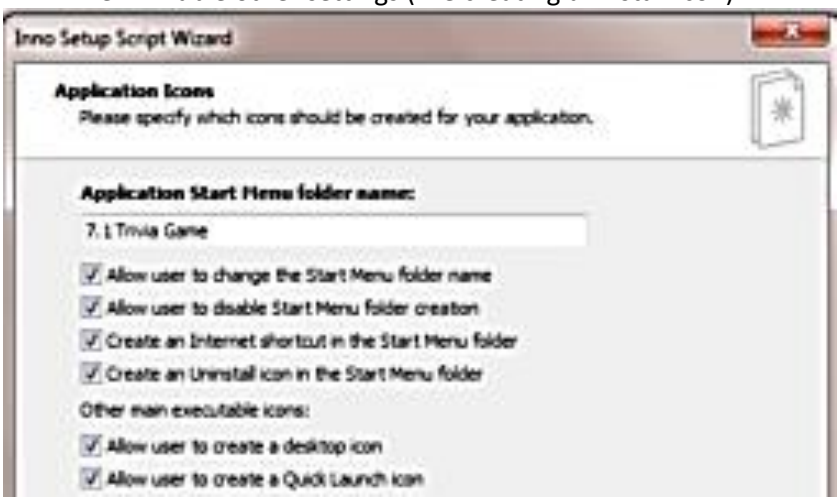
    c. Choose the location where your application will be installed on the target machine



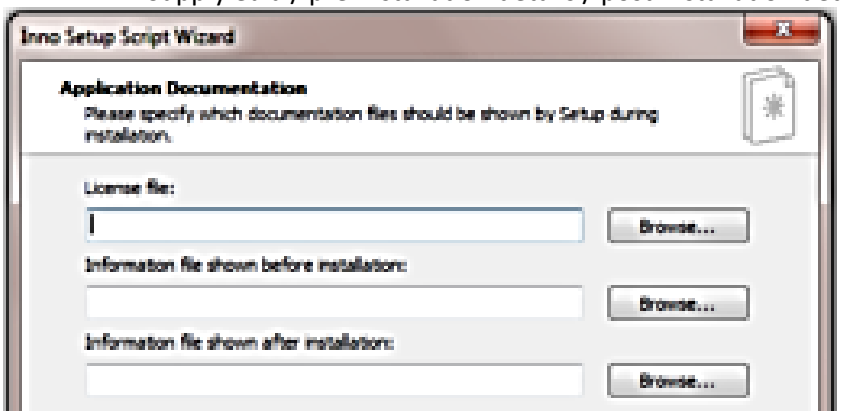
d. Browse where your compiled exe or jar is. You may also add other files and folders needed by your application here.



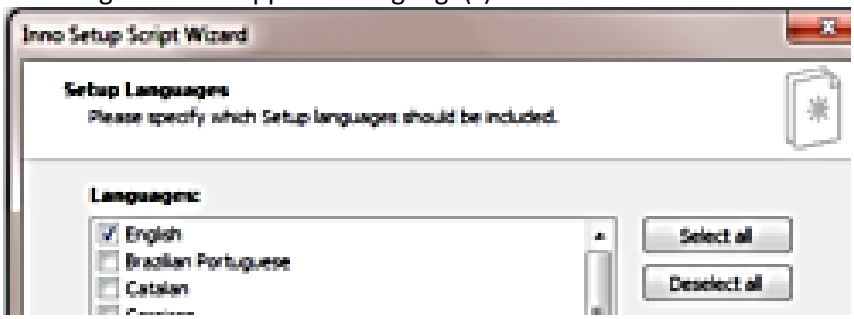
e. Enable other settings (like creating uninstall icon)



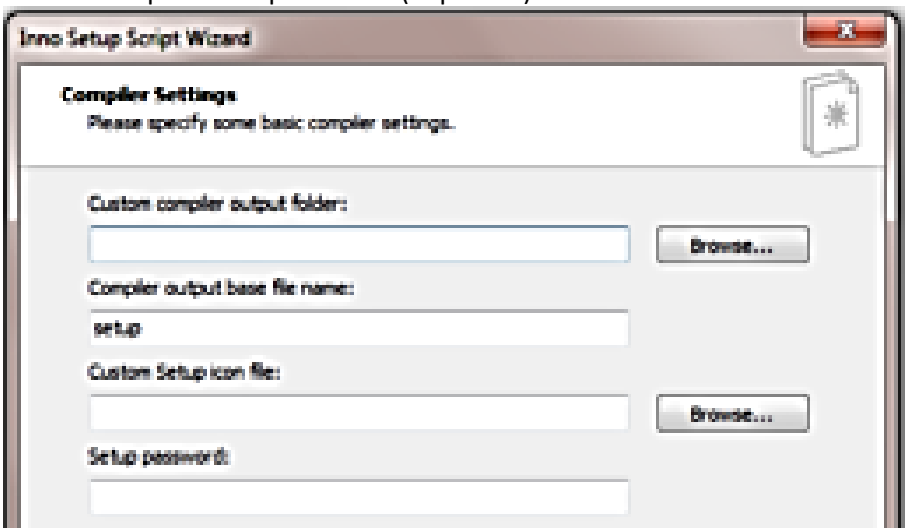
f. Supply eula / pre-installation details / post-installation details



g. Select supported language(s)



h. Select where to save the packaged installer, set it an installer name (ex. "setup"), you may also configure password protection (*optional)



Introduction to Maven

The java build tool provided by Apache to help in the build, documentation and dependency process of projects with any level of complexity written in Java and C# that uses Project Object Model (POM) and that follows the convention of source code, compiling code, and so on is called Maven. It is kind of declarative and follows the .xml file system, and also helps in dependency management of the build process with a consistent interface. Repositories are used, and it is mostly considered as a project management tool as it can manage all the project dependencies and maintenance.

Understanding of Maven

It is a project build tool that comes under the license of Apache, and there are whole hosts of libraries available in the Maven repository.

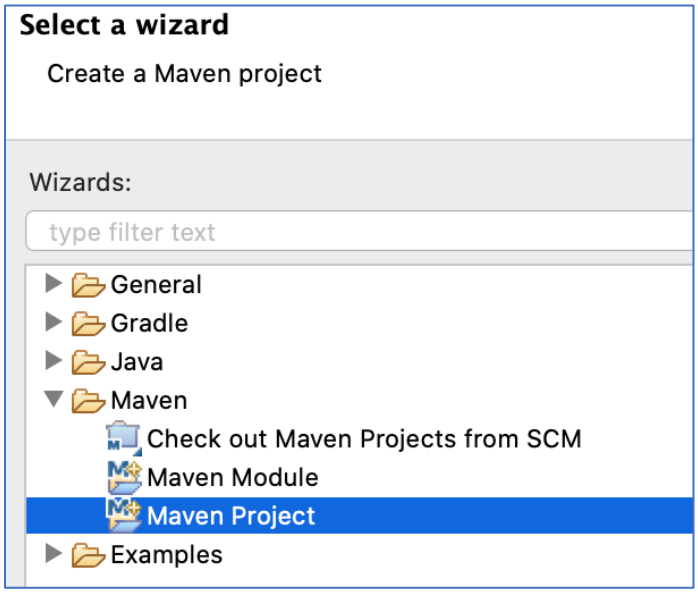
In a project, getting the right JAR files is a difficult task where there could be conflicts in the versions of the two separate packages. However, It makes sure all the JAR files are present in its repositories and avoid any such conflicting scenario. To get those JAR files from Maven, we need to visit the Maven repository and search for the exact dependencies, such as the Spring dependency, Hibernate dependency, etc.

Understanding the project is the first and foremost requirement in it. In a Maven project, the file which is of utmost importance is the pom.xml file. Now, based on the dependencies, you need to mention the exact name in this pom.xml file. For example, if you need Hibernate dependencies, you need to specify them within the dependencies tags.

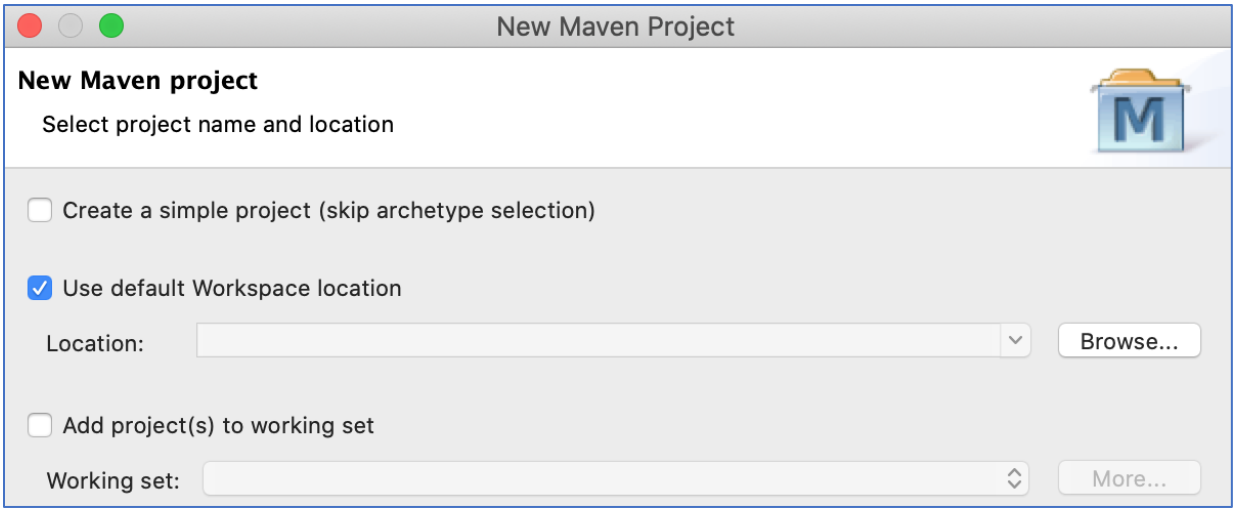
Using Maven in Eclipse IDE

1. Creating a Simple Maven Project in Eclipse

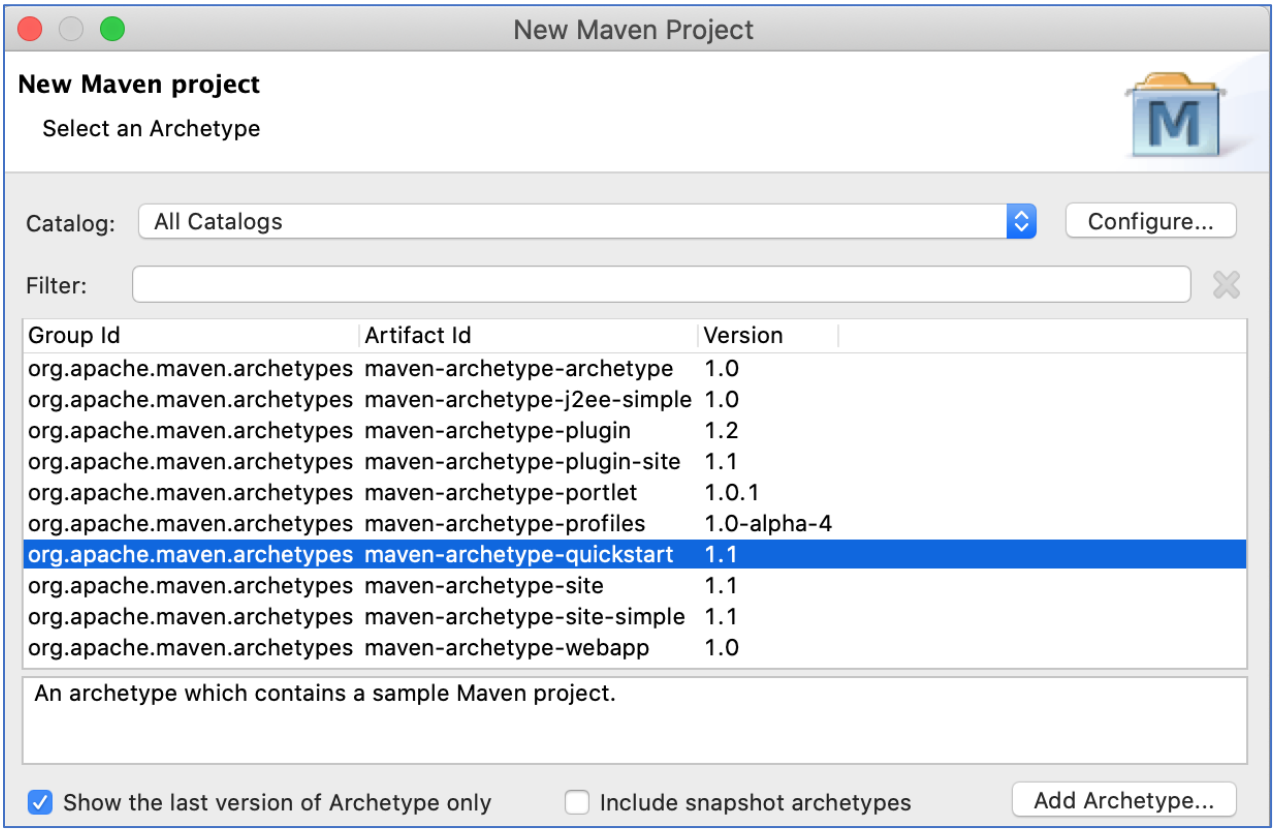
Go to File -> New -> Project and it will open the new project wizard. Select “Maven Project” as shown in the below image and click on “Next” button.



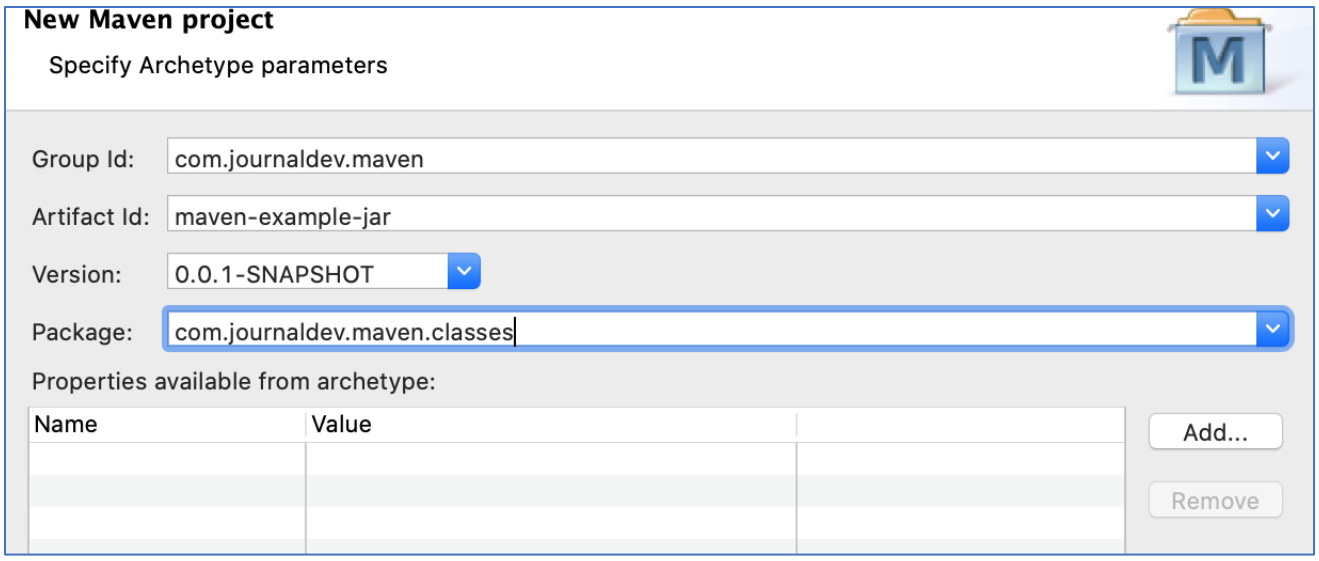
In the next screen, we have the option to specify the project location and add it to any working set. We will not make any changes to it. There is also an option to skip archetype selection and create a simple project. But, we will use the archetype to create the template project.



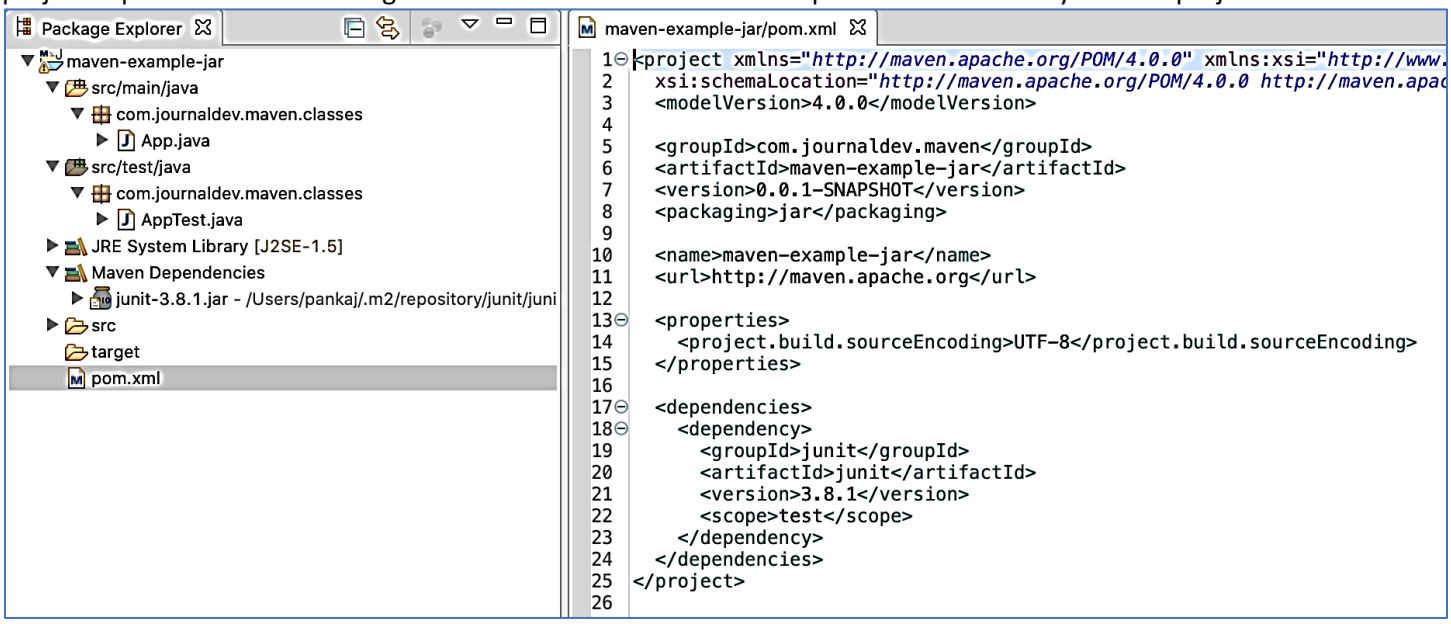
In the next screen, we have to select the archetype of the project. This is the most important step in creating the project. We want a simple maven JAR based application. So, we will choose the “maven-archetype-quickstart” artifact to create the project.



In the next screen, we have to specify the groupId, artifactId, and base package of the project. You can use the values from the image below or enter any values you want in your project.

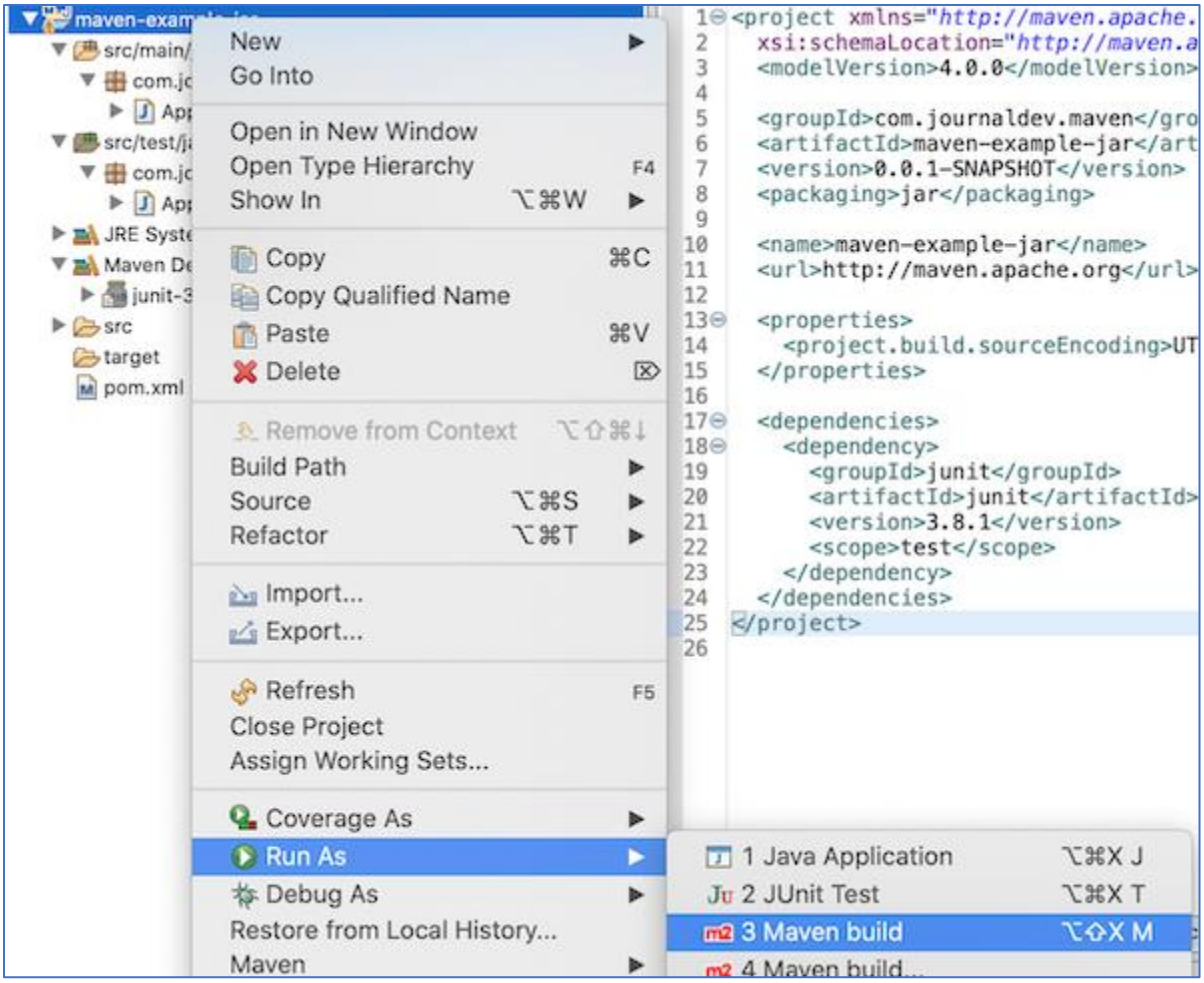


After clicking the Finish button, the popup wizard will close and the project will get created. It will be visible in the project explorer. The below image shows all the directories and the pom.xml of the newly created project.

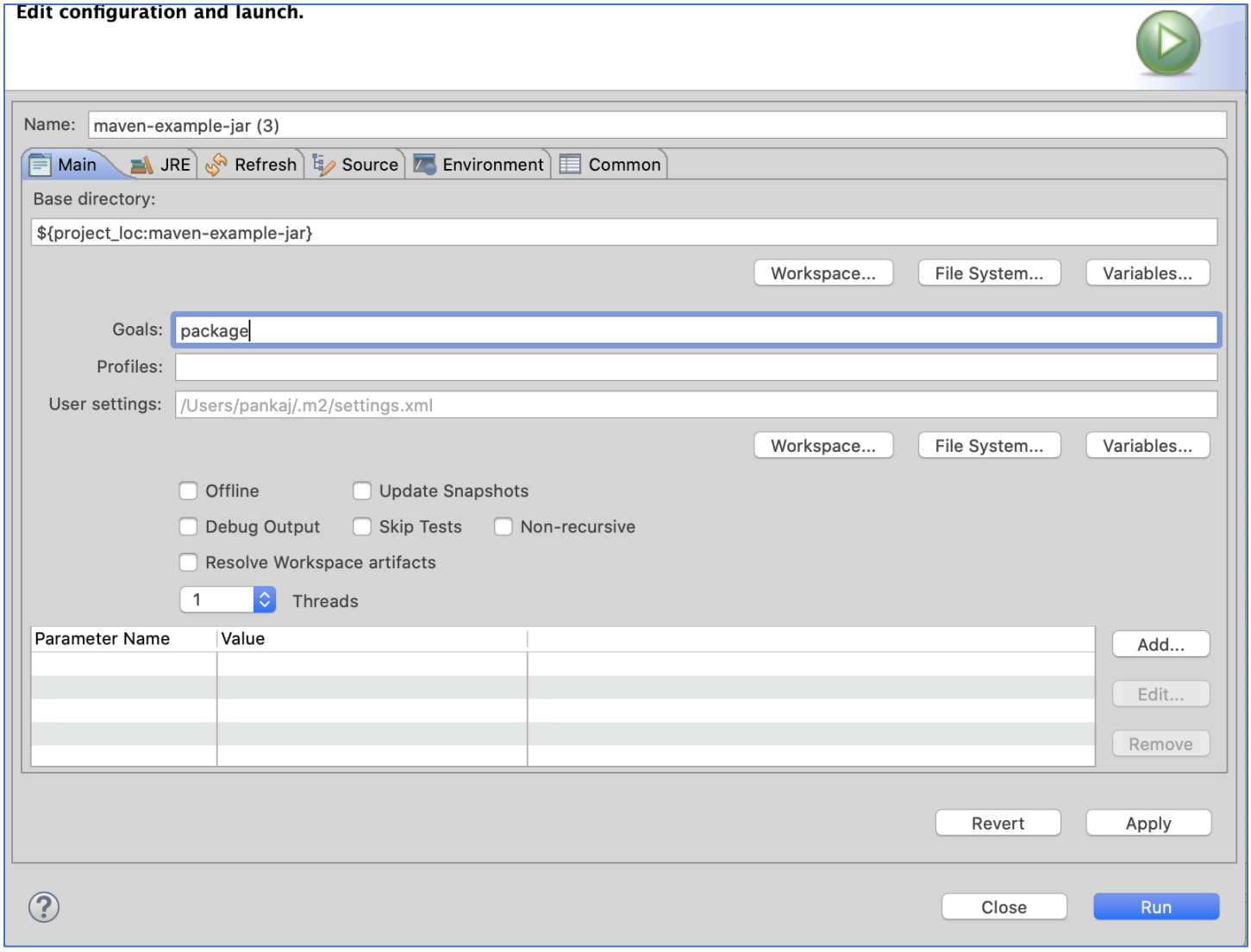


2. Building the Maven Project in Eclipse

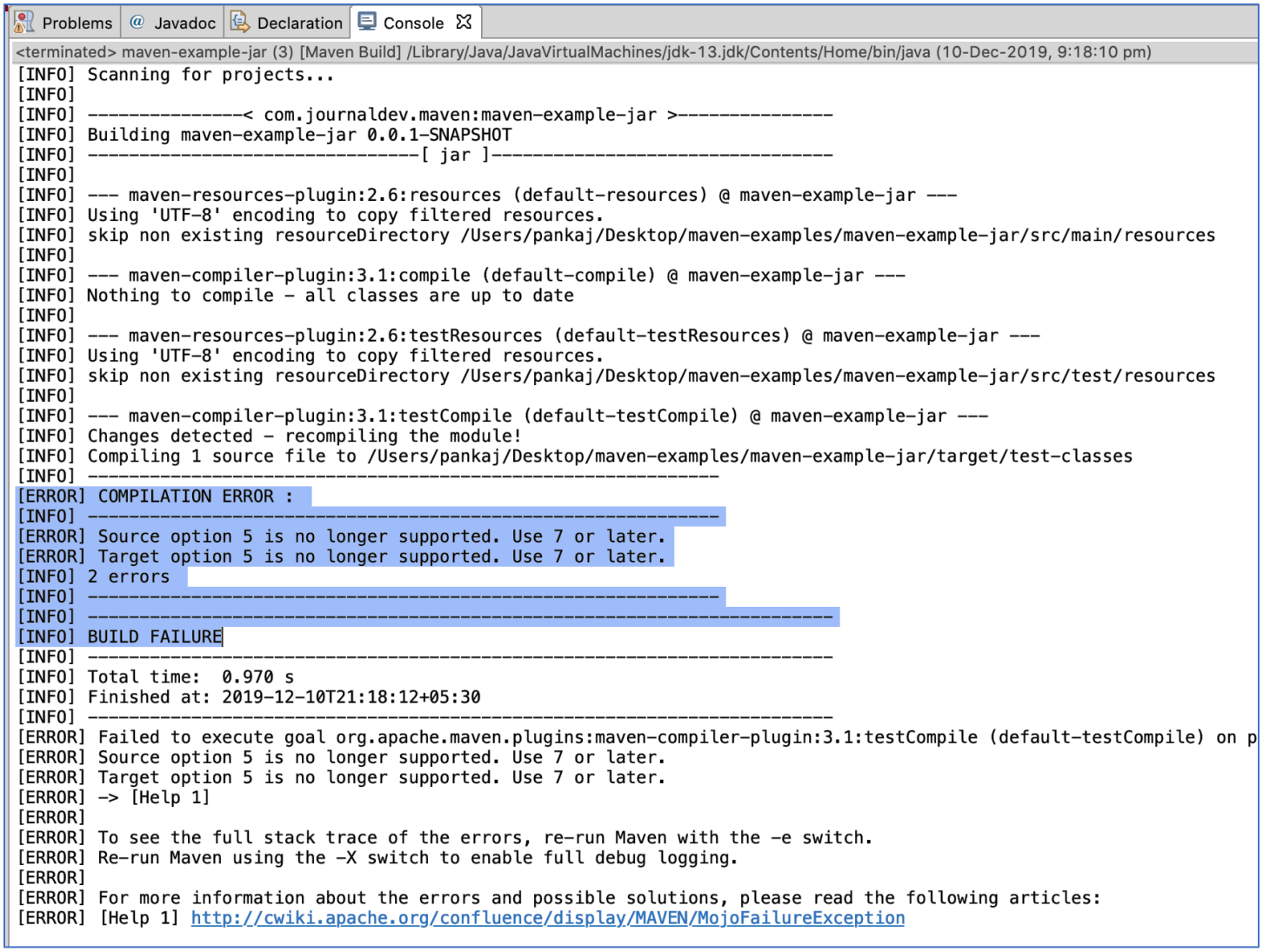
First of all, select the project and go to “Run As -> Maven Build”.



The “Edit Configuration” popup window will open. Enter the “Goals” as “package” to build the project and click on the Run button.



If you look at the Console messages, you will notice that the build failed with the compilation error messages as “Source option 5 is no longer supported. Use 7 or later”.



It's because of the auto-generated pom.xml file, which is not compatible with the latest Java versions. We can fix it by using the latest version of maven-compiler-plugin and setting maven.compiler.release to the Java version we are using.

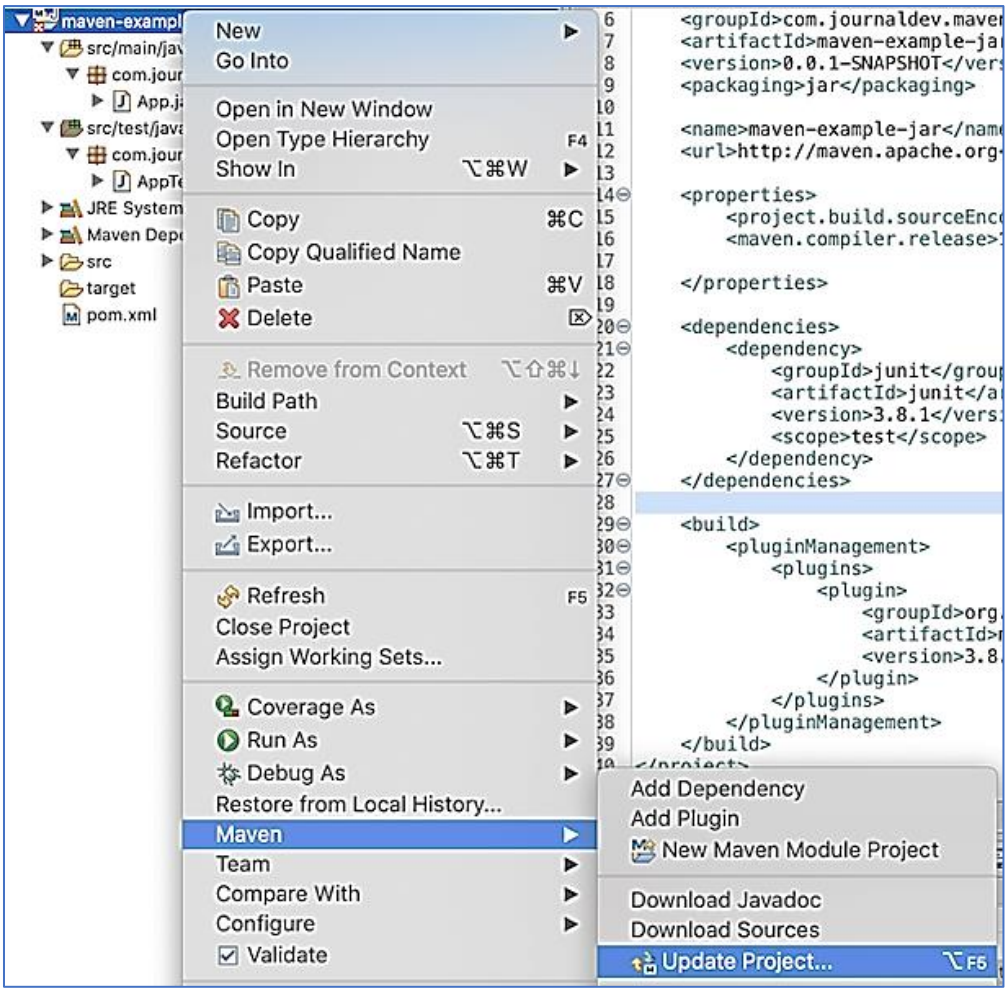
```
<properties>
	<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	<maven.compiler.release>13</maven.compiler.release>
</properties>

<build>
	<pluginManagement>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.1</version>
			</plugin>
		</plugins>
	</pluginManagement>
</build>
```

maven-example-jar/pom.xml

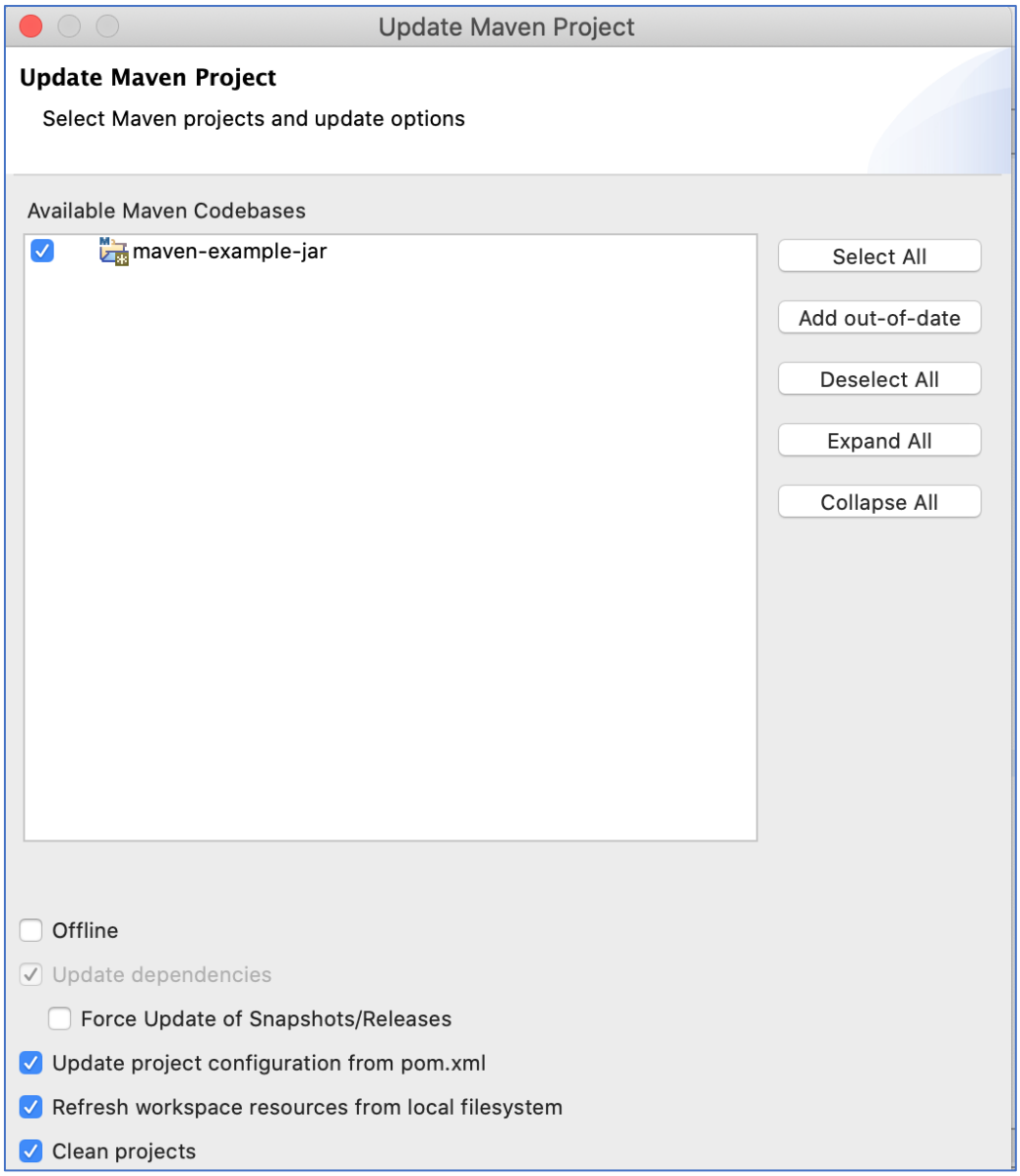
```
    <groupId>com.journaldev.maven</groupId>
    <artifactId>maven-example-jar</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>maven-example-jar</name>
    <url>http://maven.apache.org</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.release>13</maven.compiler.release>

    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>

```

3. Updating the Maven Project in Eclipse

Since we have changed the pom.xml file, we have to update the maven project to use the new configurations. You will notice a cross mark in the project root in the package explorer. The Problems view will also show the error that the project pom.xml is changed and we have to update the project. Select the project and go to “Maven > Update Project”.

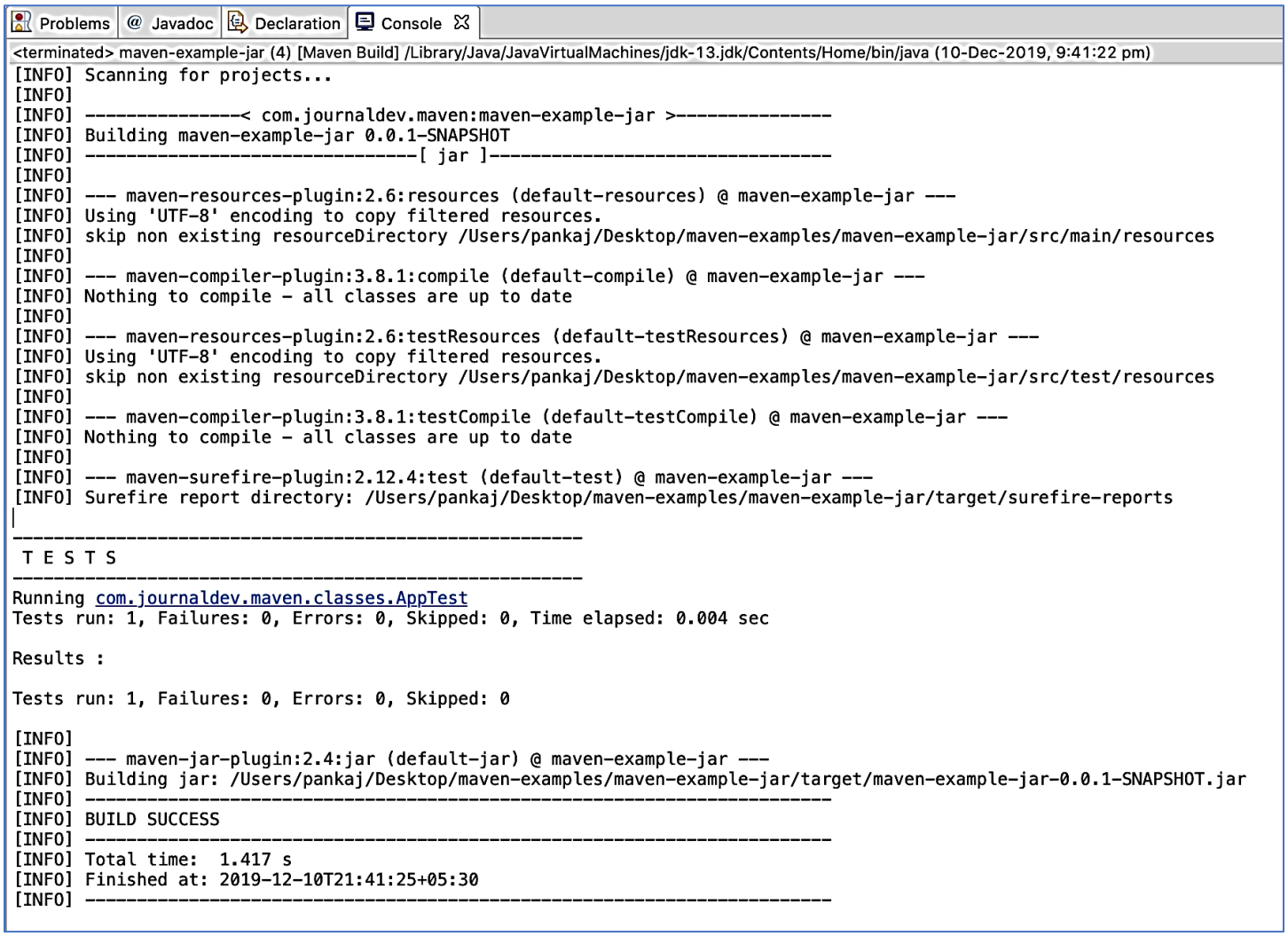


In the popup window, make sure the project is selected. Also, “Update project configuration from pom.xml” should be checked. If you want maven to download all the required dependencies again, check the “Force update of Snapshots/Releases” option. Click on the Ok button and the project will get updated with the latest pom.xml configurations.

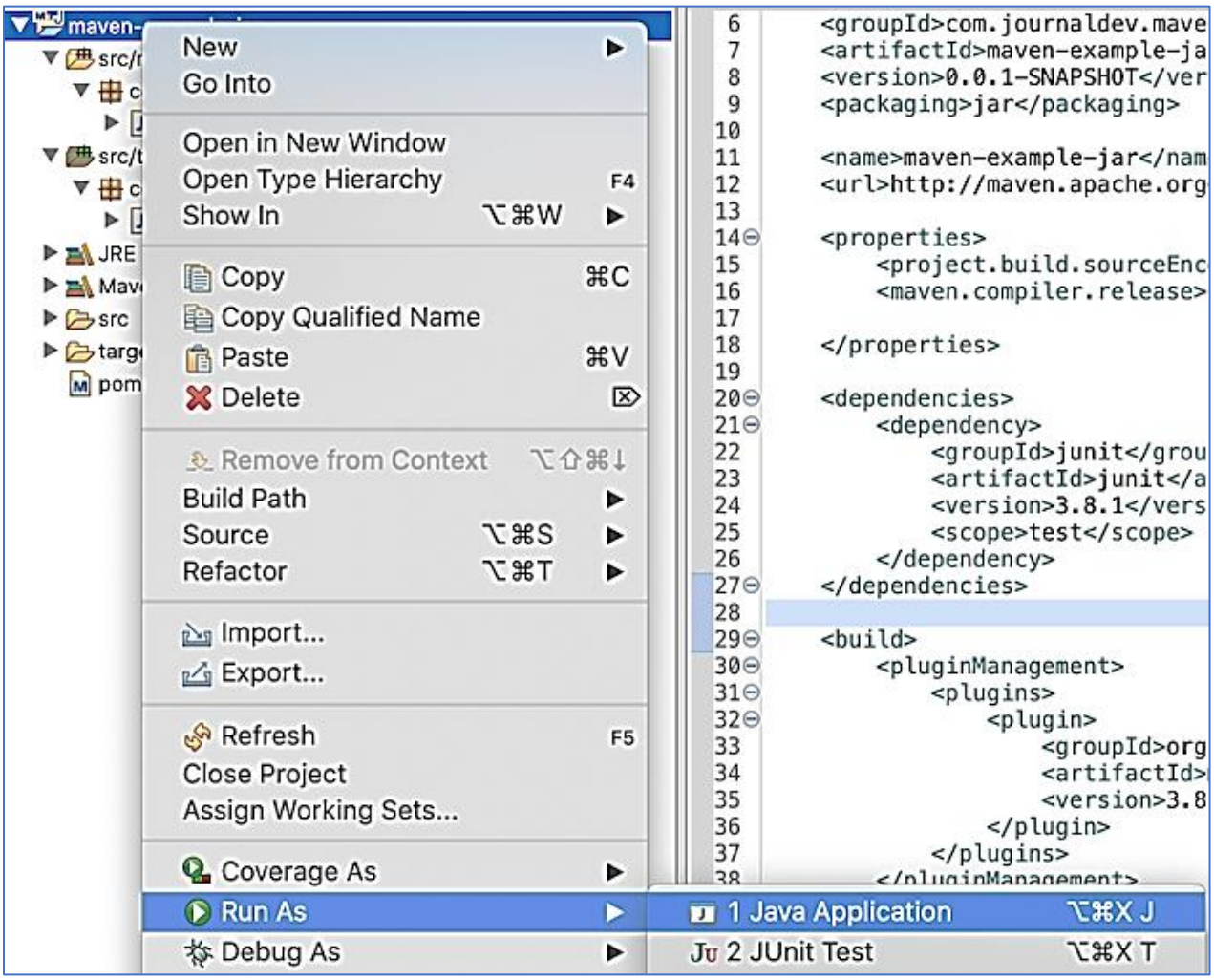


4. Building and Running the Maven Project in Eclipse

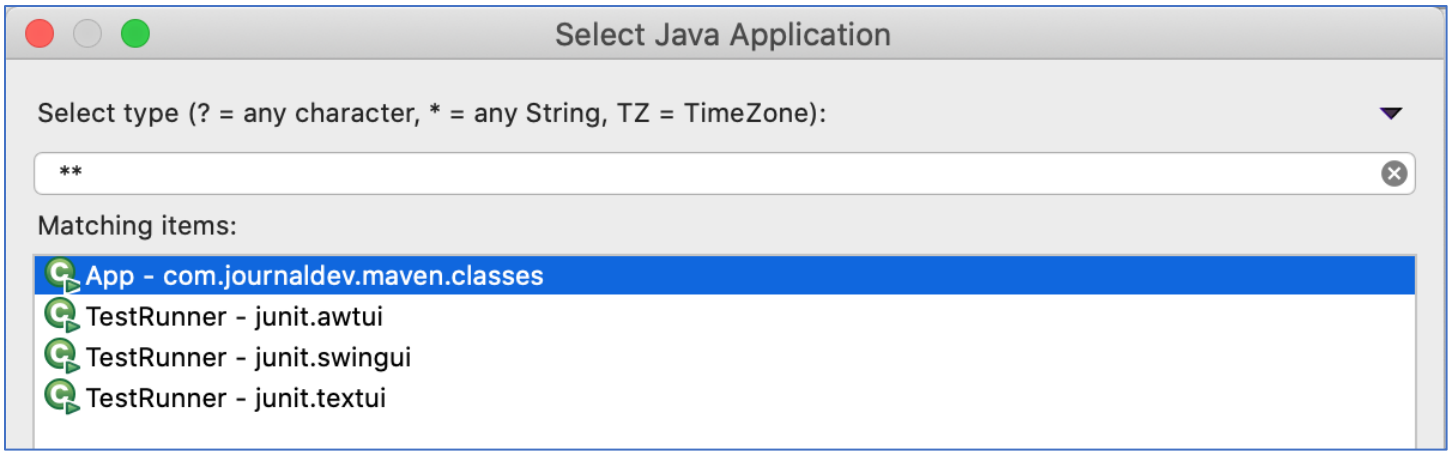
Now that we have fixed the pom.xml configurations, build the project once again as shown above. This time the Console should show the “BUILD SUCCESS” message.



To run the maven project, select it and go to “Run As > Java Application”.



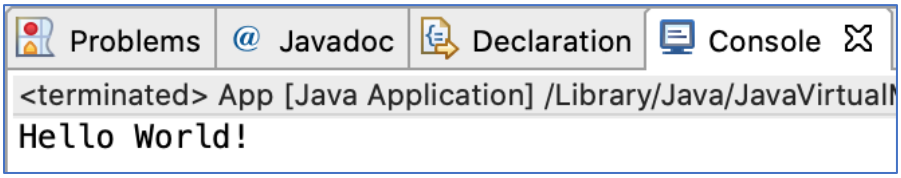
In the next window, select the main class to execute. In this case, select the App class and click on the Ok button.



The App class code is:

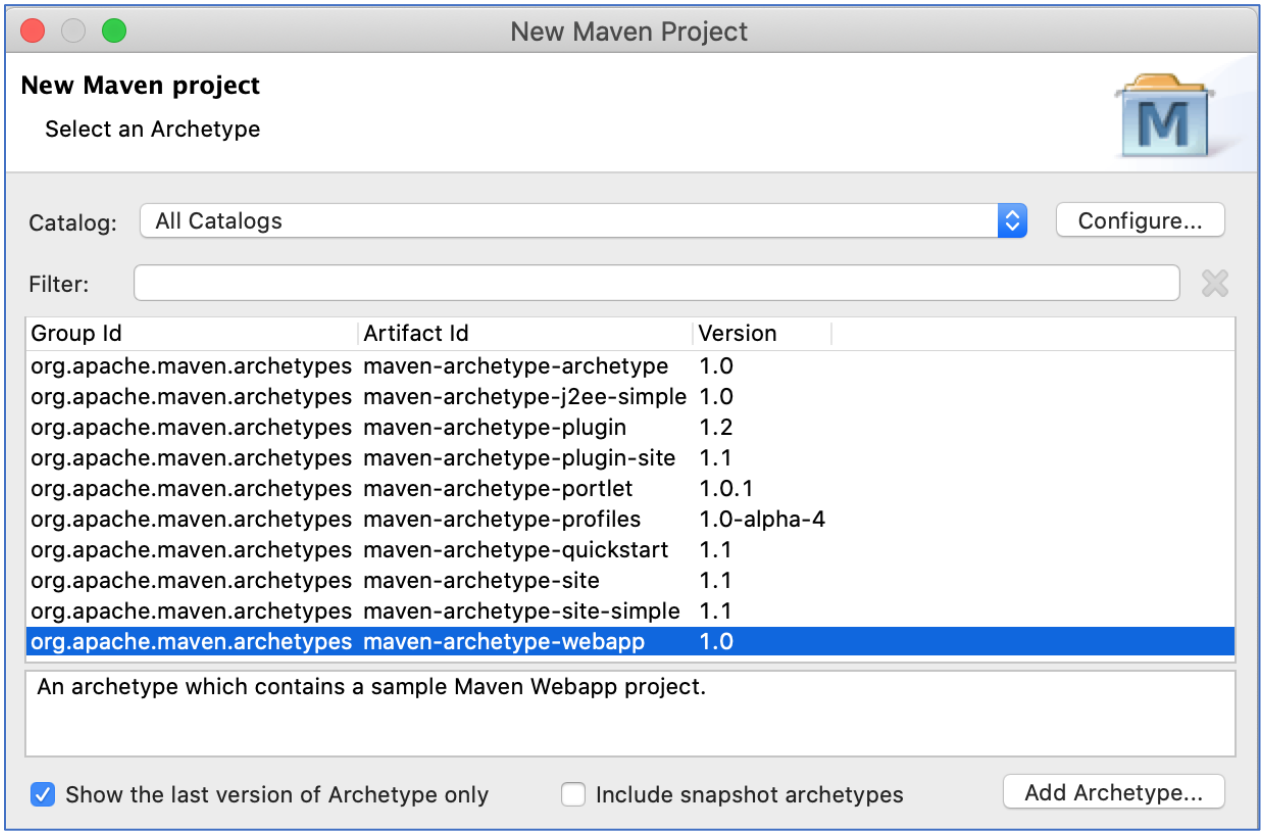
```
package com.journaldev.classes;

public class App {

	public static void main(String[] args) {

		System.out.println("Hello World");
	}

}
```

You will see the “Hello World” output in the Console window.



5. Creating a Java Web Project in Eclipse using Maven

The process of creating a java web project using Maven is almost the same. The only change is that we will use “maven-archetype-webapp” so that a web application project is created.

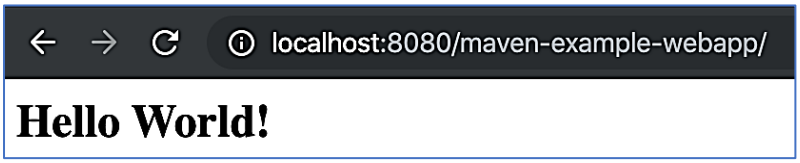


The next step would be to fix the pom.xml file by using the latest version of the maven-compiler-plugin and using the latest java version. Then build the project with the “package” goal and you should see the “BUILD SUCCESSFUL” message in the console.

Problems | Javadoc | Declaration | Console

```
<terminated> maven-example-webapp [Maven Build] /Library/Java/JavaVirtualMachines/jdk-13.jdk/Contents/Home/bin/java (10-Dec-2019, 9:53:21 pm)
[INFO] Scanning for projects...
[INFO]
[INFO] -------------< com.journaldev.maven:maven-example-webapp >--------------
[INFO] Building maven-example-webapp Maven Webapp 0.0.1-SNAPSHOT
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ maven-example-webapp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ maven-example-webapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ maven-example-webapp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/pankaj/Desktop/maven-examples/maven-example-webapp/src/test/resour
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ maven-example-webapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ maven-example-webapp ---
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ maven-example-webapp ---
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.thoughtworks.xstream.core.util.Fields (file:/Users/pankaj/.m2/repository/c
WARNING: Please consider reporting this to the maintainers of com.thoughtworks.xstream.core.util.Fields
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Packaging webapp
[INFO] Assembling webapp [maven-example-webapp] in [/Users/pankaj/Desktop/maven-examples/maven-example-webapp/target,
[INFO] Processing war project
[INFO] Copying webapp resources [/Users/pankaj/Desktop/maven-examples/maven-example-webapp/src/main/webapp]
[INFO] Webapp assembled in [27 msecs]
[INFO] Building war: /Users/pankaj/Desktop/maven-examples/maven-example-webapp/target/maven-example-webapp.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.238 s
[INFO] Finished at: 2019-12-10T21:53:24+05:30
[INFO] ------------------------------------------------------------------------
```

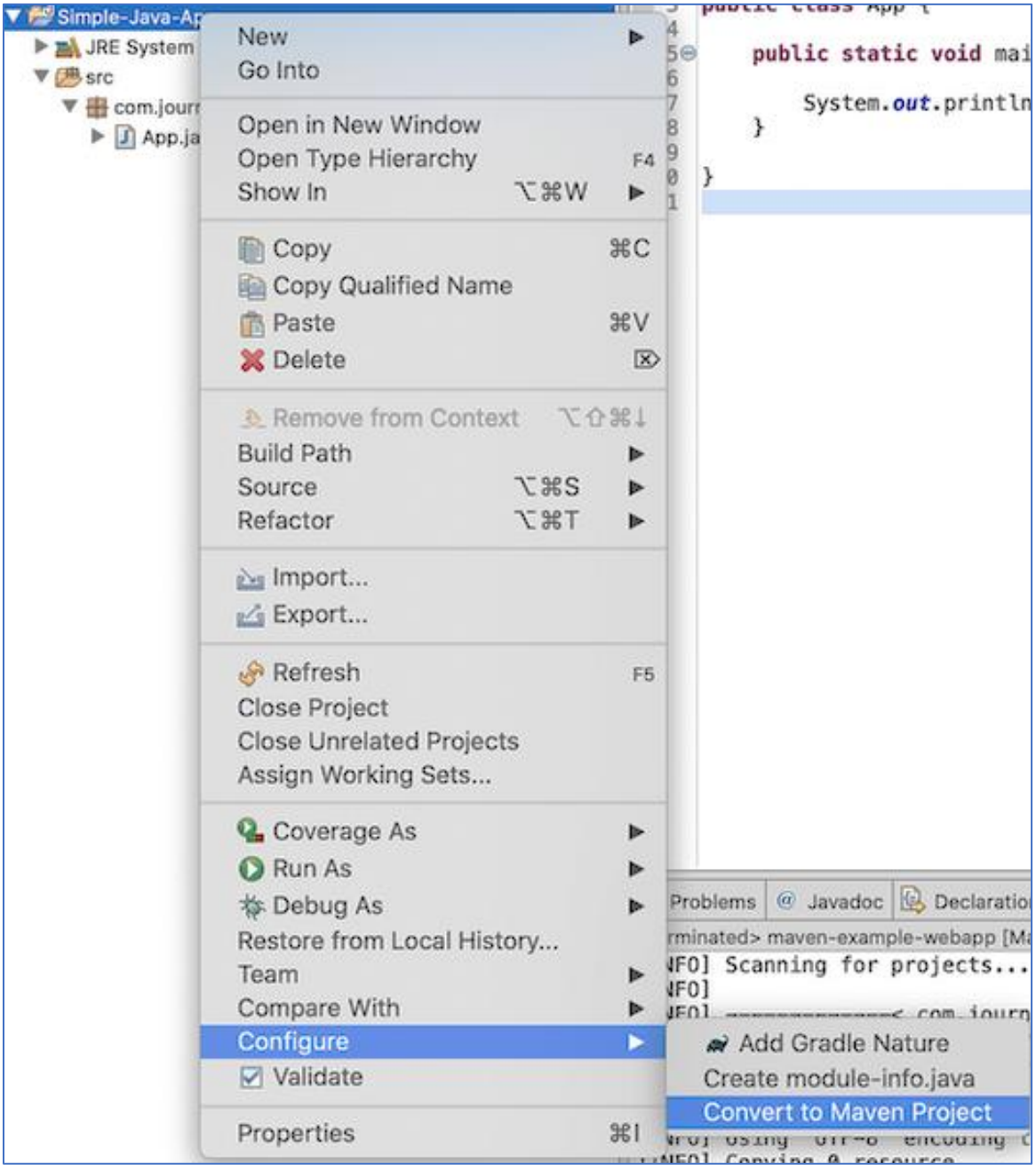
Notice the project build generates maven-example-webapp.war file in the target directory. We can integrate Apache Tomcat in the Eclipse itself, but that is out of the scope of this tutorial. You can deploy the WAR file into any servlet container and you should see the following output.



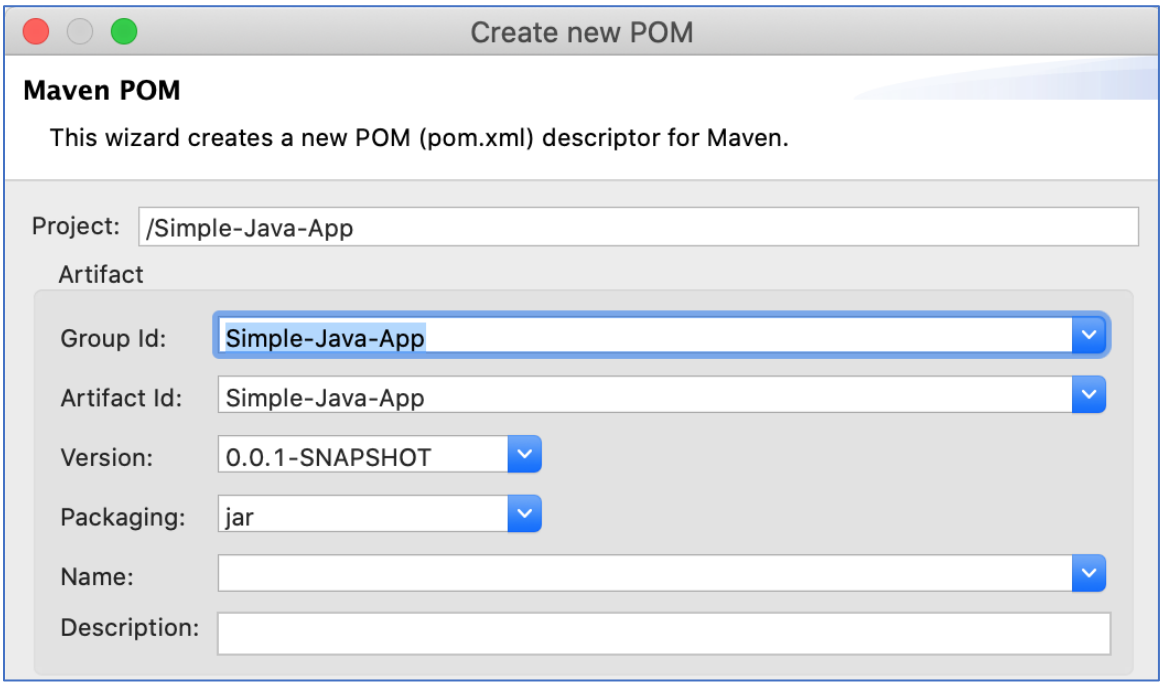
6. Converting a simple Java project to maven

Sometimes we have a simple Java project. It's recommended to use Maven for dependency and build management. We can easily convert a simple Java project to a maven based project. Let's say we have a simple Java project like the below image.

Select the project and go to "Configure > Convert to Maven Project".



It will open up a new window to create a pom.xml file. We have to provide project configurations such as groupId, artifactId, packaging, etc.



The below image shows the maven project structure and the newly generated pom.xml file.