

香港中文大學(深圳)

The Chinese University of Hong Kong

(Materials of this lecture are NOT included in the midterm and final exams)

CSC3100 Data Structures

Lecture 2: A brief introduction to Java

Li Jiang
School of Data Science (SDS)
The Chinese University of Hong Kong, Shenzhen



Outline

- ▶ Why do we use Java to learn data structures?
- ▶ What will we learn and NOT learn about Java?
- ▶ Basic knowledge of Java
 - JDK/JVM/JRE
 - Keywords, declaration, expressions, class/object, method, others



Why do we choose Java?

- ▶ This course is not just about reading and writing — **we need to write codes by implementing data structures and algorithms**
- ▶ Java is one of the most frequently used programming languages
 - **It is one of the easy-to-use programming languages to learn**
 - Java is platform-independent, i.e., **write once, run anywhere!**
- ▶ Used for
 - Developing Android Apps
 - Helps you to create Enterprise Software
 - Wide range of mobile applications
 - Scientific computing
 - **Big data analytics (e.g., Hadoop and Spark)**
 - ...
 - **Much more!**



Java vs C/C++

▶ Advantages of Java

- Easier to learn
- No pointer, safer
- Automatic memory management, including garbage collection
- Cross platforms
- More powerful standard libraries
- Java and Java-based IDEs are often provided free of charge
- Often used for research (e.g., Hadoop and Spark)
- ...



What we will learn about Java?

► What will we learn?

- JDK, JVM, JRE
- **Keywords**
- **Simple declarations**
- **Statements**
- **Classes/objects**
- **Methods**
- **Exceptions**

► What will we **NOT** learn?

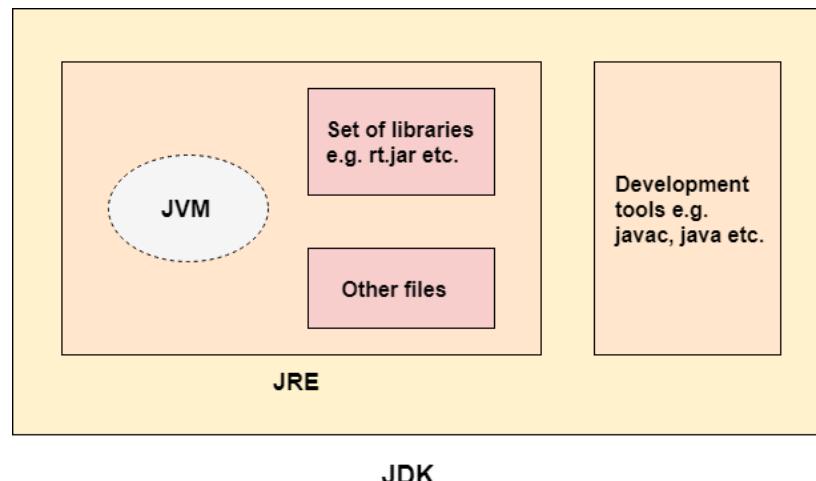
- Interface
- Abstract class
- Inheritance
- GUI
- Multi-thread
- Garbage collection
- Network communication
- Web design
- Android Apps development
- ...
- Many others

Only focus on the basic knowledge of Java that is used for implementing the data structures and algorithms in this course!



JDK, JVM, JRE

- ▶ Java Development Kit (JDK)
 - A software development environment which is used to develop Java applications and applets
- ▶ Java Runtime Environment (JRE)
 - It provides the minimum requirements for executing a Java application; it consists of JVM, core classes, etc.
- ▶ JVM (Java Virtual Machine)
 - An abstract machine that doesn't physically exist, a specification that provides a runtime environment in which Java bytecode can be executed





OOP (object-oriented programming)

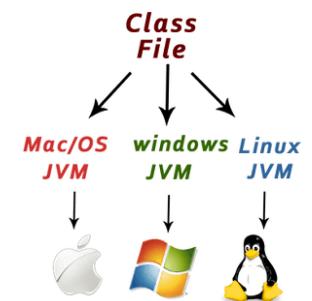
- ▶ Everything in Java is an object
 - We organize our software as a combination of different types of objects that incorporate both **data** and **behavior**
 - Class “Object” is the root class for **every** derived class in Java

- ▶ OOP features:
 - Object
 - Class
 - Inheritance
 - Polymorphism
 - Abstraction
 - Encapsulation



Platform independence

- ▶ Java: **write once, run anywhere**
 - Different from C/C++/..., which are compiled into platform specific machines
- ▶ Java is a software platform that runs on top of hardware platforms
 - Runtime environment
 - API (Application Programming Interface)
- ▶ Java codes can be executed on Windows, Linux, Mac/OS, etc.
 - Java program is compiled into **bytecode**
 - The bytecode runs on multiple platforms





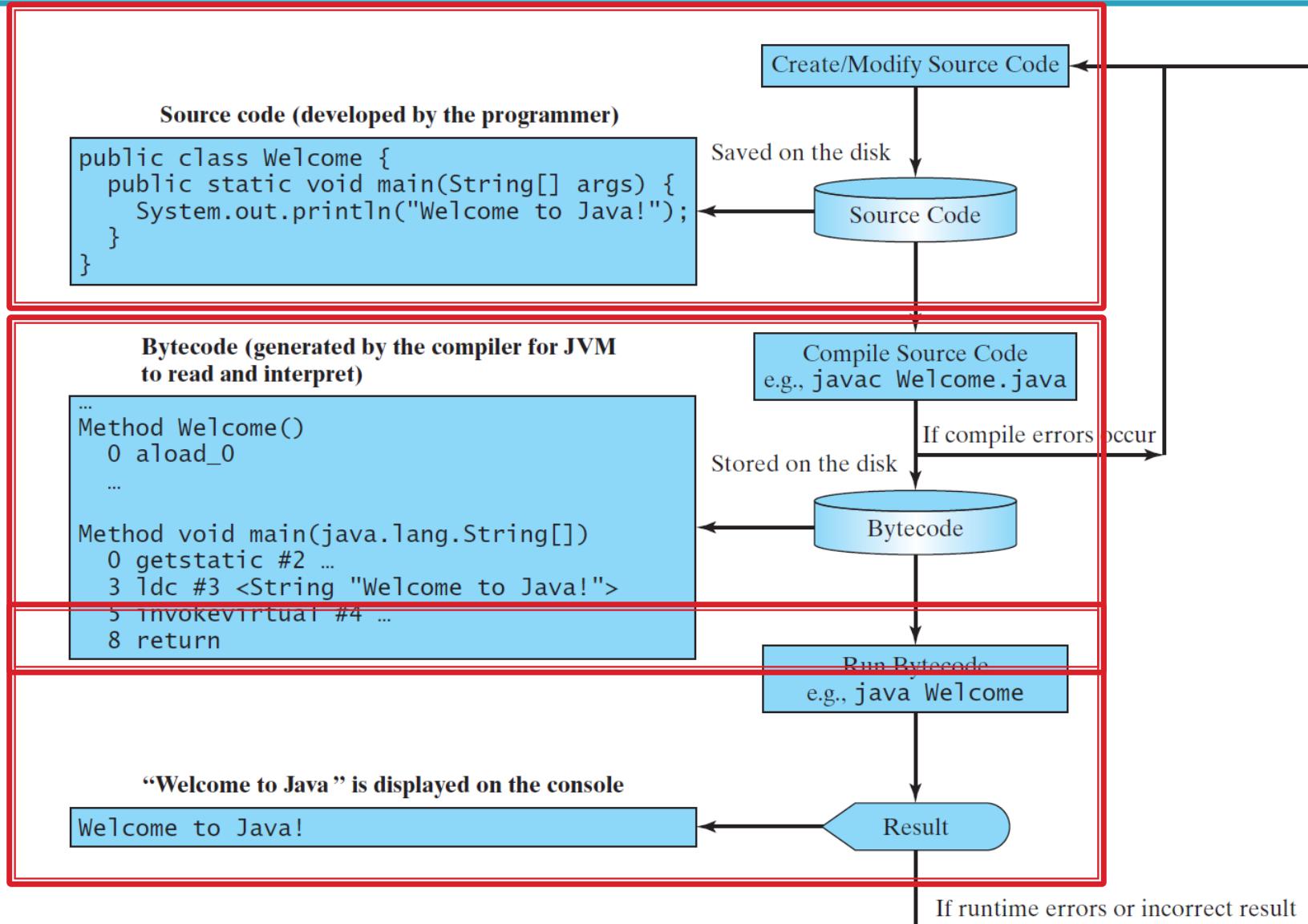
The first Java program

```
1 class Welcome {  
2     public static void main(String[] args) {  
3         System.out.println("Welcome to Java!");  
4     }  
5 }
```

- ▶ Source codes
 - Declare a class with name
 - `class Welcome{...}`
 - Declare the main method
 - `public static void main(String args[]) {...}`
 - Java main method is the entry point of any java program
 - Print "Welcome to Java!" to the console
 - `System.out.println("Welcome to Java!")`



Running the first Java program





Environment setup

- ▶ Install a JDK (Java Development Kit)
 - Install an OpenJDK
 - e.g., [Java SE DK 16](#)
 - Set up environment variables:
 - JAVA_HOME: Points to the base of the JDK installation. e.g., "C:\Program Files\Java\jdk____VERSION"
 - PATH: Add "%JAVA_HOME%\bin".
 - Verify installation (e.g., on Mac)

```
(base) TianshudeMacBook-Pro:~ tianshuyu$ java -version
java version "14.0.2" 2020-07-14
Java(TM) SE Runtime Environment (build 14.0.2+12-46)
Java HotSpot(TM) 64-Bit Server VM (build 14.0.2+12-46, mixed mode, sharing)
```

- ▶ Install an IDE or editor
 - Eclipse (my personal choice), VS Code, Notepad...
- ▶ A concise guide: <https://www.runoob.com/java/java-tutorial.html>



- ▶ Lab
- ▶ Lab1
- ▶ lab5
- ▶ Lab7
- ▶ test
- ▶ test1
- ▶ test2

New ⌘N ▶

Open File...

Open Projects from File System...

Recent Files ▶

Close Editor ⌘W

Close All Editors ⌘⌘W

Save ⌘S

Save As... ⌘S

Save All ⌘⌘S

Revert

Move...

Rename... F2

Refresh F5

Convert Line Delimiters To ▶

Print... ⌘P

Import... ⌘I

Export...

Properties ⌘I

Switch Workspace ▶

Restart

Java Project

Project...

Package

Class

Interface

Enum

Record

Annotation

Source Folder

Java Working Set

Folder

File

Untitled Text File

JUnit Test Case

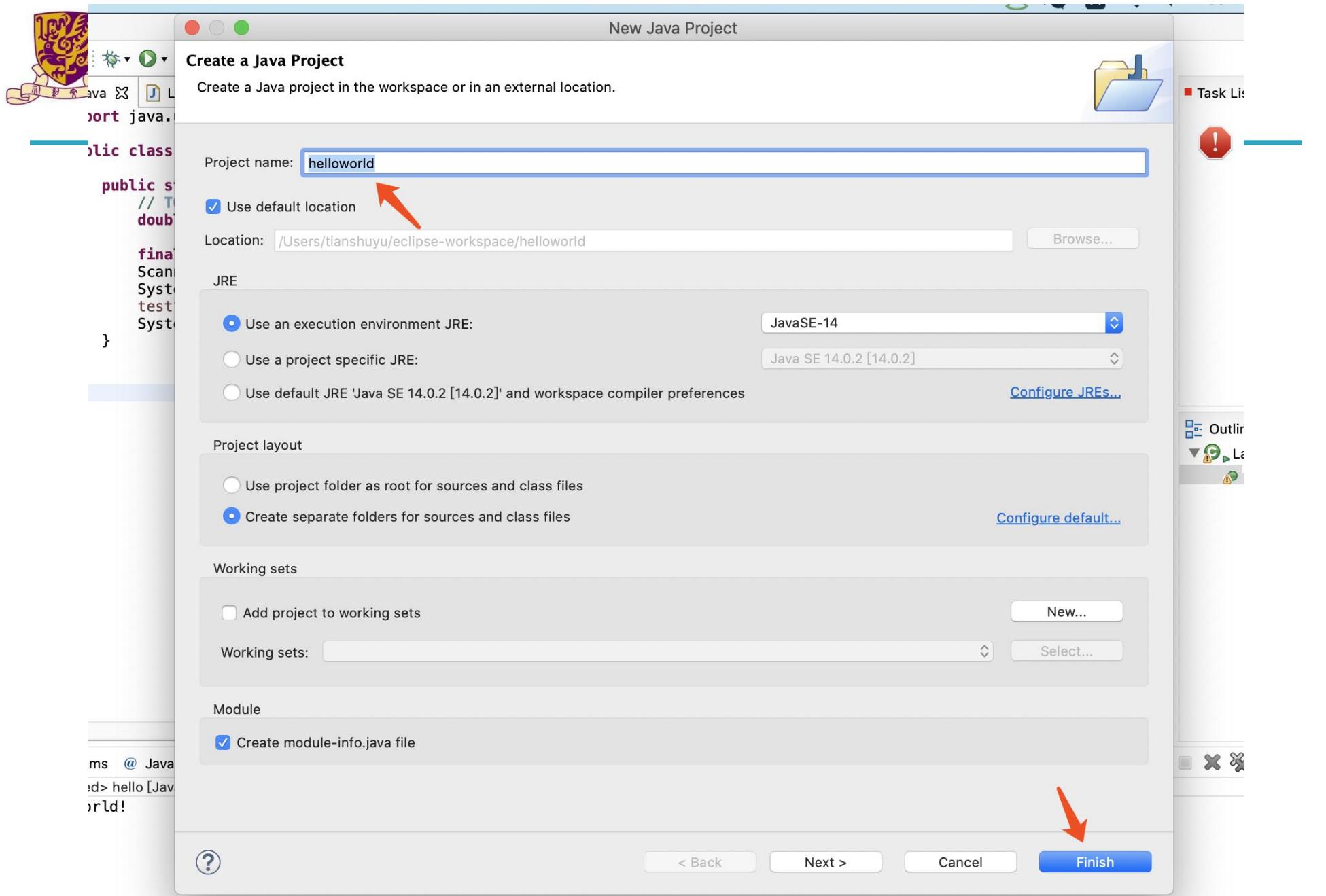
Example...

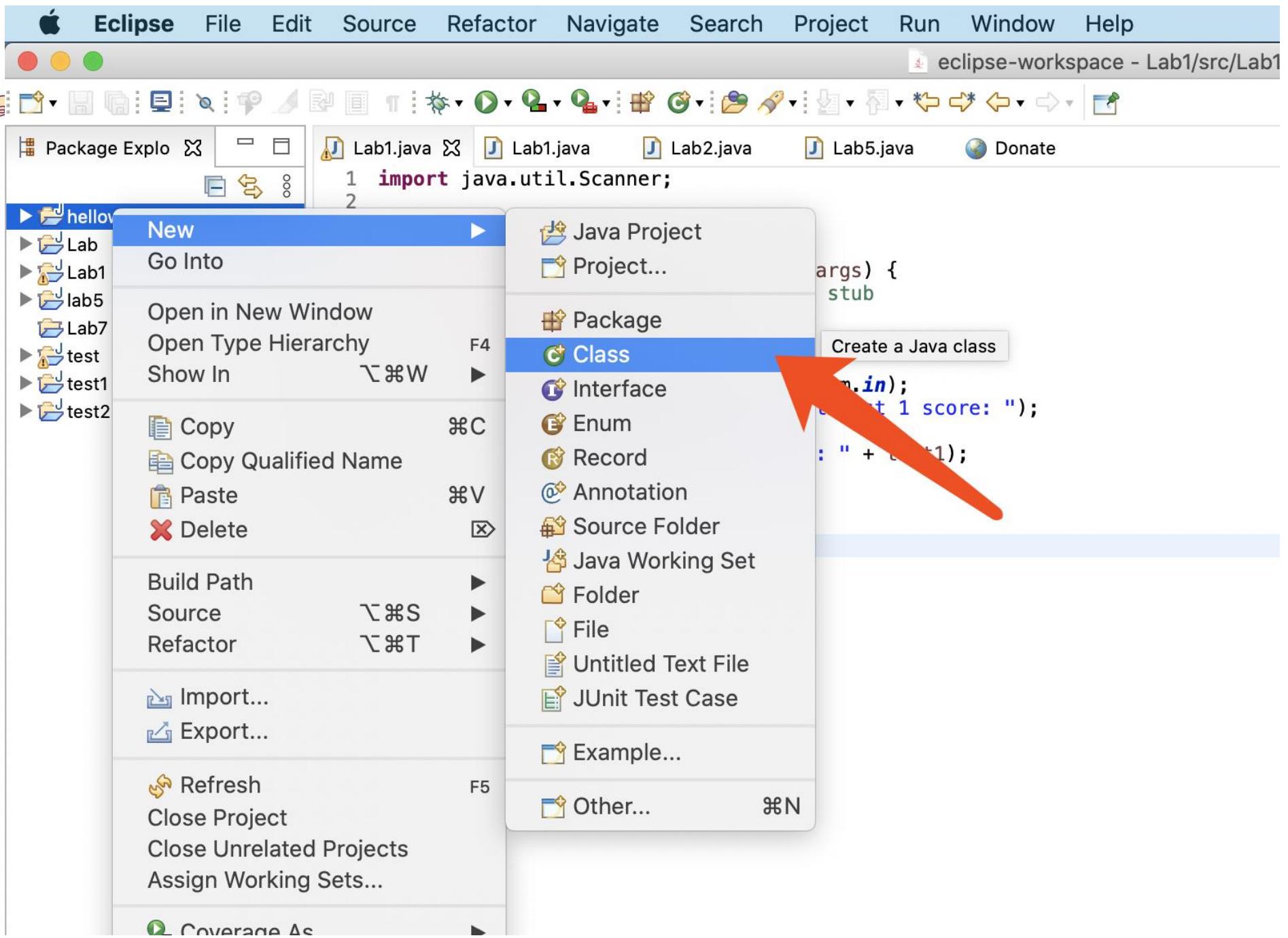
Other... ⌘N

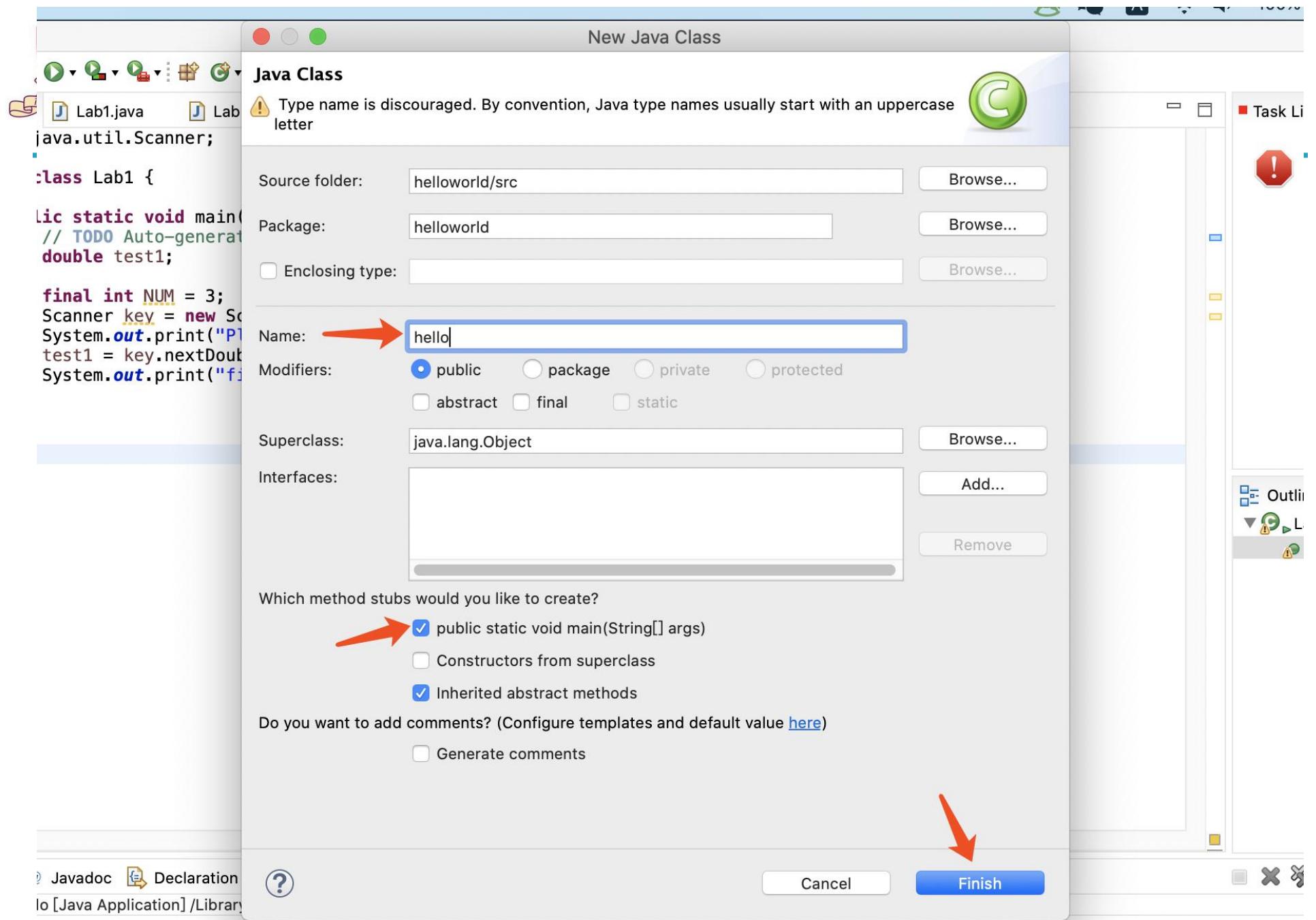
Create a Java project /src/Lab1.java

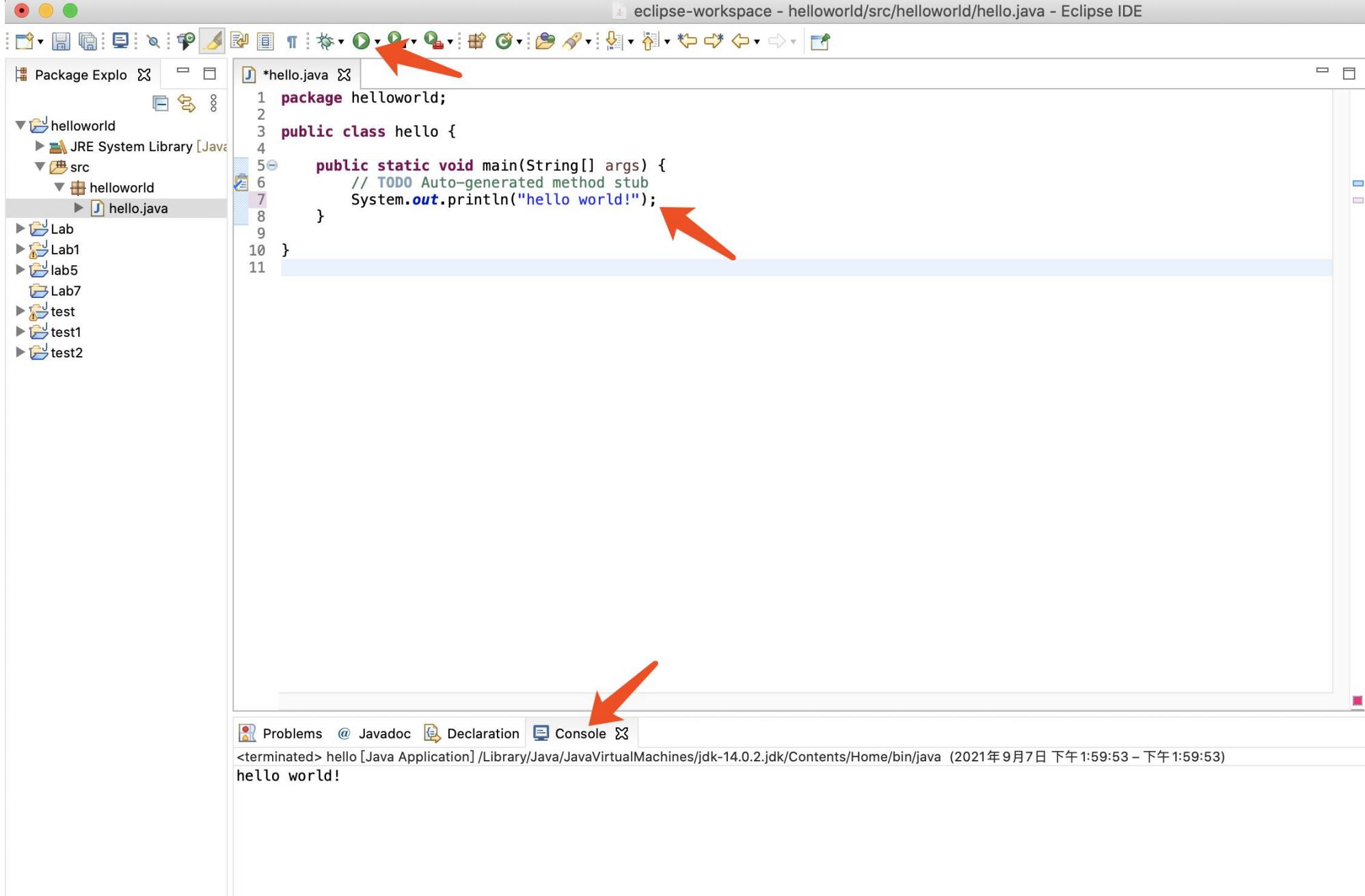


Donate











(1) Java keywords

- ▶ Java keywords (reserved words)
 - Keywords are particular words, which acts as a key to a code
 - Keywords **cannot be used as variable or object names**

- ▶ Keywords of primitive types
 - **int**: used to declare a variable that can hold a 32-bit signed integer
 - **boolean**: used to declare a variable as a boolean type (true or false)
 - **double**: used to declare a variable that can hold a 64-bit floating-point numbers
 - **char**: used to declare a variable that can hold unsigned 16-bit Unicode characters
 - **short**: used to declare a variable that can hold a 16-bit integer
 - **long**: used to declare a variable that can hold a 64-bit integer
 - **float**: used to declare a variable that can hold a 32-bit floating-point number
 - **byte**: used to declare a variable that can hold an 8-bit data values



(1) Java keywords

- ▶ **Keywords of loops**
 - **for**: used to create a for loop (a variable initialization, a boolean expression, and an incrementation) `for (int x = 0; x < 10; x = x + 1)`
 - **while**: used to create a while loop, which tests a boolean expression and executes some statements if the expression is true `while (x < 10)`
 - **continue**: used to continue the loop; it continues the current flow of the program and skips the remaining code at the specified condition
 - **break**: used to break loop or switch statement; it breaks the current flow of the program at specified condition
- ▶ **Keywords of conditions**
 - **if**: Java if keyword tests the condition, and the if block is executed if the condition is true
 - **else**: used to indicate the alternative branches in an if statement
- ▶ **Keywords of classes and objects**
 - **class**: used to declare a class
 - **new**: used to create new objects



(1) Java keywords

▶ Keywords of exceptions

- **try**: used to start a block of code that will be tested for exceptions, and the try block must be followed by either catch or finally block
- **catch**: used to catch the exceptions generated by try statements, and it must be used after the try block only
- **finally**: used to create a block of code following a try block; its block always executes whether an exception occurs or not

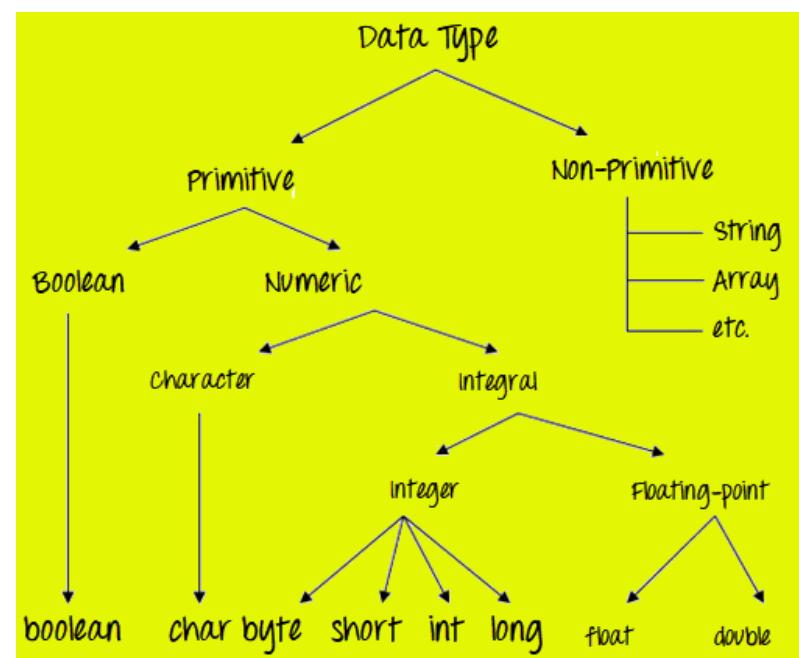
▶ Keywords of others

- **import**: used to make classes and interfaces available and accessible to the current source code
- **package**: used to declare a Java package that includes the classes
- **public**: as an access modifier, it is used to indicate that an item is accessible anywhere, and it has the widest scope among all other modifiers
- **return**: used to return from a method when its execution is complete
- **null**: used to indicate that a reference does not refer to anything. It removes the garbage value



(2) Simple declaration

- ▶ Variable in Java is a data container, storing the data values during program execution
- ▶ Every variable is assigned a data type which designates the type and quantity of value it can hold
- ▶ Variable is a memory location name of the data
- ▶ To use variables
 - Variable declaration
 - Variable initialization





(2) Simple declaration

▶ Variable declaration

```
int m, n;      // Two integer variables  
double x, y;   // Two real coordinates  
boolean b;     // Either 'true' or 'false'  
char ch;        // A character, such as 'P' or '@'
```

type

variable
name

semicolon



(2) Simple declaration

- ▶ Numeric expressions are written in much the same way as in other languages

```
n = 3 * (5 + 2);
x = y / 3.141592653;
n = m % 8;      // Modulo, i.e. n is now (m mod 8)
b = true;
ch = 'x';
```

- ▶ Division operator has two different things:

```
double f;
f = 1 / 3;           // f is now 0.0
f = 1.0 / 3.0;       // f is now 0.33333333...
```



(2) Simple declaration

```
class Guru99 {  
    static int a = 1; //static variable  
    int data = 99; //instance variable  
    void method() {  
        int b = 90; //local variable  
    }  
}
```

- ▶ Three types of variables
 - Local variables are declared inside the body of a method
 - Instance variables are defined without STATIC keyword, and they are defined outside a method declaration, i.e., they are object specific
 - Static variables are initialized only once, at the start of the program execution; These variables should be initialized first, before the initialization of any instance variables, i.e., they are Class specific



(2) Simple declaration

- ▶ Type conversion (casting)
 - Assign a real value to an integer value: need a cast

```
double radians;  
int degrees;  
...  
degrees = radians * 180 / 3.141592653;           // Error  
degrees = (int) (radians * 180 / 3.141592653); // OK
```

- Assigning an integer to a real variable does not need casting



(2) Simple declaration

- ▶ A string is a sequence of characters, or an array of characters

```
//String is an array of characters
char[] arrSample = {'R', 'O', 'S', 'E'};
String strSample_1 = new String (arrSample);
```

- ▶ Use **String** class to handle strings

```
package codes;
import java.lang.String;

public class StringMethods {

    public static void main(String[] args) {

        String str1 = "Software";
        String str2 = "Testing";
        System.out.println(str1 + str2);
        System.out.println(str1.concat(str2));
    }
}
```

Output:

```
Problems @ Javadoc Declaration Console ×
<terminated> StringMethods [Java Application] C:\Program Files\Java\
SoftwareTesting
SoftwareTesting
```



(3) Statements

- ▶ Statements can be grouped in blocks using “{}”
- ▶ If and if-else statements

```
if (n == 3)  
    x = 3.2;
```

Note:

- There is no then keyword
- The condition must be of boolean type and written within parentheses
- Comparison is made using ‘==’

- ▶ Comparison operators: >, <, ==, >=, <=, !=

```
if (x != 0)  
    y = 3.0 / x;           // Executed when x is non-zero  
else  
    y = 1;                 // Executed when x is zero
```



(3) Statements

► More about Boolean expressions

<i>and</i>	&&
<i>or</i>	
<i>not</i>	!

```
int x, y;
boolean b;
...
if ((x <= 9 || y > 3) && !b) {
    b = true;
}
```

► **while** loop statement: need the stopping criterion

```
// Calculate exp(1). End when the term is less than 0.00001
double sum = 0.0;
double term = 1.0;
int k = 1;
while (term >= 0.00001) {
    sum = sum + term;
    term = term / k;
    k++; // Shortcut for 'k = k + 1'
}
```



(3) Statements

▶ for loop statement

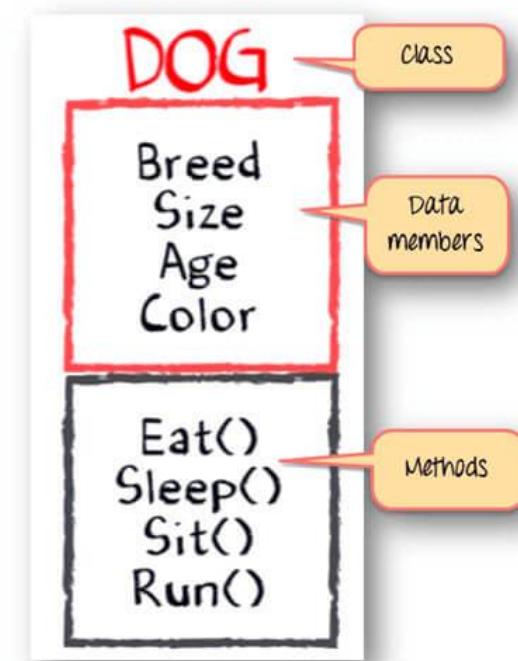
- Initial value of value i
- Stopping criterion of the loop
- How to change the value of i in each iteration

```
// Calculate 1 + (1/2) + (1/3) + ... + (1/100)
int i;
double sum = 0.0;
for (i = 1; i <= 100; i++) {
    sum = sum + 1.0 / i;
}
```



(4) Class/object

- ▶ A **class** is a blueprint or prototype that defines the variables and the methods (functions) common to all Java objects of a certain kind
- ▶ An **object** is a specimen of a class
 - An object is an instance of a class
 - Software objects are often used to model real-world objects you find in everyday life





(4) Class

- ▶ A class declaration contains
 - A set of attributes (called **instance variables**)
 - A set of functions (called **methods** in Java)

```
class Turtle {  
    private boolean penDown;  
    protected int x, y;  
  
    // Declare some more stuff  
}
```

- ▶ Methods in Java
 - Main methods: **public static void main(String [] args){...}**
 - Constructor methods: **public Turtle(){...}**
 - General methods: **public void jumpTo(int newX, int newY) {...}**



(4) Class

▶ Java constructor

- A special method for initializing a newly created object and is called just after the memory is allocated for the object
- It can be used to initialize the objects to desired values or default values at the time of object creation
- It is not mandatory to write a constructor for a class

▶ Rules for creating a java constructor

- It has the **same name** as the class
- It **should not** return a value (not even void)



(4) Class

▶ Java method

- Method name, input parameters, method body, return type

```
class Turtle {  
    // Attribute declarations, as above  
  
    public void jumpTo(int newX, int newY) {  
        x = newX;  
        y = newY;  
    }  
  
    public int getX() {  
        return x;  
    }  
}
```

```
public Turtle(int initX, int initY) {  
    x = initX;  
    y = initY;  
    penDown = false;  
}
```



(4) Class

Access modifiers

- The **public** keyword is used to declare that something can be accessed from other classes
- The **protected** keyword specifies that something can be accessed from within the class and all its subclasses, but not from the outside
- When we do not mention any access modifier, it is called default access modifier, and the scope of this modifier is limited to the package only
- The **private** declaration means that those attributes cannot be accessed outside of the class
 - In general, attributes should be kept private

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No



(4) Class

- ▶ In Java, statements can only be written within methods in classes
 - There must be some method which is called by the system when the program starts executing, which is **main** method

```
01. public class HelloWorld {  
02.     public static void main(String[] args) {  
03.         System.out.println("Hello World!");  
04.     }  
05. }
```

- ▶ Notes
 - **static** keyword: when the main method is called, it is not associated with an object, but with the class
 - The parameter **args**: if the Java interpreter is given any more information than the class name, this data is passed on to the main method in this parameter



(4) Class

- The keyword **new** is used to create an object of a class

```
Turtle t;  
t = new Turtle(100, 100);
```

- Calling methods in objects

```
int a = t.getX();  
t.jumpTo(300, 200);
```

- Java has garbage collection, so no need to destroy objects manually

```
public class ConfunDemo3 {  
    public static void main(String[] args){  
        Person z=new Person("zhangsan",3);  
        z.show();  
    }  
}  
  
class Person{  
    private String name;  
    private int age;  
    public Person(String n,int m){  
        name=n;  
        age=m;  
    }  
    //getter  
    public String getName(){  
        return name;  
    }  
    public int getAge(){  
        return age;  
    }  
    public void show(){  
        System.out.println(name+"\n"+age);  
    }  
}
```



(5) Exception

- ▶ Many things can go wrong during the execution of a program (programmers may introduce faults)
 - E.g., division by zero or calling a method with a null reference
- ▶ Throw exceptions
 - E.g., consider a method to read a positive integer from the keyboard; what if the input character is not an integer?

```
public int getNatural() throws IOException {  
    char ch;  
    while (more input) {  
        ch = (read character);  
        if (ch < '0' || ch > '9') {  
            throw new IOException("bad natural number");  
        }  
        ...  
    }  
    ...  
}
```



(5) Exception

▶ Catch exceptions

- The statement(s) within the try clause are executed as usual, but whenever an exception occurs, the try clause is interrupted and the statements within the corresponding catch clause are executed

```
int m, n;
try {
    n = getNatural();
    m = n * 2; // If an exception is thrown, this is not executed
}
catch (IOException e) {
    // The user entered something wrong. Use 1 as default.
    n = 1;
    m = 2;
}
```



(6) Others

- ▶ Print something: writing to the console
 - `System.out.print(xxx)`
 - `System.out.println(xxx)`

```
System.out.print("Jag vill bo ");  
System.out.println("i en svamp");  
System.out.println("Annars får jag kramp");
```

The resulting output is:

```
Jag vill bo i en svamp  
Annars får jag kramp
```

Variable values can be printed like this:

```
int a;  
a = 6 * 7;  
System.out.println("6 * 7 = " + a);
```



(6) Others

▶ Packages

- Package in Java is a collection of classes, sub-packages, and interfaces
- It helps organize your classes into a folder structure and make it easy to locate and use them
- More importantly, it helps improve code reusability

```
import java.awt.*;
```



(6) Others

▶ Comments

- Single sentence: two forward slashes //
- A block of codes: /* xxx */

```
1 import java.util.*;
2
3 /**
4  * This program demonstrates object construction.
5  * @version 1.01 2004-02-19
6  * @author Cay Horstmann
7  */
8 public class ConstructorTest
9 {
10    public static void main(String[] args)
11    {
12        // fill the staff array with three Employee objects
13        Employee[] staff = new Employee[3],
14
15        staff[0] = new Employee("Harry", 40000);
16        staff[1] = new Employee(60000);
17        staff[2] = new Employee();
18
19        // print out information about all Employee objects
20        for (Employee e : staff)
21            System.out.println("name=" + e.getName() + ",id=" + e.getId() + ",salary="
22                           + e.getSalary());
23    }
24 }
```



(6) Others

▶ Array

- Declaration

```
int[] someInts;           // An integer array  
Turtle[] turtleFarm;    // An array of references to Turtles
```

- Initialization

```
someInts = new int[30];  
turtleFarm = new Turtle[100];
```

- Use arrays: 0 ~ size-1

```
int i;  
for (i = 0; i < someInts.length; i = i + 1) {  
    someInts[i] = i * i;  
}
```

What is the maximum array size?



(6) Others

- ▶ **ArrayList** is a data structure that can
 - be stretched to accommodate additional elements within itself
 - shrink back to a smaller size when elements are removed

```
ArrayList<Object> a = new ArrayList<Object>();  
add(Object o);  
  
remove(Object o);
```

- ▶ Notes
 - A key data structure for handling dynamic behaviors of elements
 - When the number of elements is larger than its initial size, it creates another larger interval array and then copied the existing elements to the new interval array
 - Although it provides more flexibility, it often take more space cost than an array



(6) Others

▶ Keyword “this”

- **this** keyword in Java is a reference variable that refers to the current object of a method or a constructor
- The main purpose of using **this** keyword is to remove the confusion between class attributes and parameters that have same names

eclipse-workspace1 - OnlineProgramming/src/JournalDev/Item.java - Eclipse

```
File Edit Source Refactor Navigate Search Project Run Window Help  
Item.java  
1 package JournalDev;  
2  
3 public class Item{  
4     String name;  
5  
6     // Constructor with a parameter  
7     public Item(String name) {  
8         name = name;  
9     }  
10  
11     // Call the constructor  
12     public static void main(String[] args) {  
13         Item Obj = new Item("car");  
14         System.out.println(Obj.name);  
15     }  
16 }  
17
```

eclipse-workspace1 - OnlineProgramming/src/JournalDev/Item.java - Eclipse

```
File Edit Source Refactor Navigate Search Project Run Window Help  
Item.java  
1 package JournalDev;  
2  
3 public class Item{  
4     String name;  
5  
6     // Constructor with a parameter  
7     public Item(String name) {  
8         this.name = name;  
9     }  
10  
11     // Call the constructor  
12     public static void main(String[] args) {  
13         Item Obj = new Item("car");  
14         System.out.println(Obj.name);  
15     }  
16 }  
17
```

Console
<terminated> Item
null

eclipse-workspace1 - OnlineProgramming/src/JournalDev/Item.java - Eclipse

```
File Edit Source Refactor Navigate Search Project Run Window Help  
Item.java  
1 package JournalDev;  
2  
3 public class Item{  
4     String name;  
5  
6     // Constructor with a parameter  
7     public Item(String name) {  
8         this.name = name;  
9     }  
10  
11     // Call the constructor  
12     public static void main(String[] args) {  
13         Item Obj = new Item("car");  
14         System.out.println(Obj.name);  
15     }  
16 }  
17
```

Console
<terminated> Item
car



(6) Others

eclipse-workspace - CSC3100/src/javaIO/MyIOClass.java - Eclipse

Quick Access

Package Explorer

CSC3100
JRE System Library [JavaSE-
src
javaIO
MyIOClass.java

MyIOClass.java

```
1 package javaIO;
2 import java.io.BufferedReader;
3
4 public class MyIOClass {
5
6     public static void main(String[] args) {
7         String inFilePath = "/Users/yxfang/Documents/eclipse-workspace/inputInfo";
8         String outFilePath = "/Users/yxfang/Desktop/eclipse-workspace/outputInfo";
9         try {
10             FileReader fileReader = new FileReader(inFilePath);
11             BufferedReader stdin = new BufferedReader(fileReader);
12
13             FileWriter fileWriter = new FileWriter(outFilePath);
14             BufferedWriter stdout = new BufferedWriter(fileWriter);
15
16             String line = null;
17             while((line = stdin.readLine()) != null) {
18                 String s[] = line.split(" ");
19                 int a = Integer.parseInt(s[0]);
20                 int b = Integer.parseInt(s[1]);
21
22                 int sum = a + b;
23                 String sumStr = String.valueOf(sum);
24                 stdout.write(sumStr);
25                 stdout.newLine();
26             }
27
28             stdout.flush();
29             stdout.close();
30             stdin.close();
31         }catch(IOException e) {
32             System.out.println("something wrong");
33             e.printStackTrace();
34             System.exit(0);
35         }
36     }
37 }
38 }
```

inputInfo

1	1	3
2	2	4
3	5	6
4		

outputInfo

1	4
2	6
3	11
4	

Problems @ Javadoc Declaration Console

<terminated> MyIOClass [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home/bin/java (2021年6月4日 下午3:06:24)

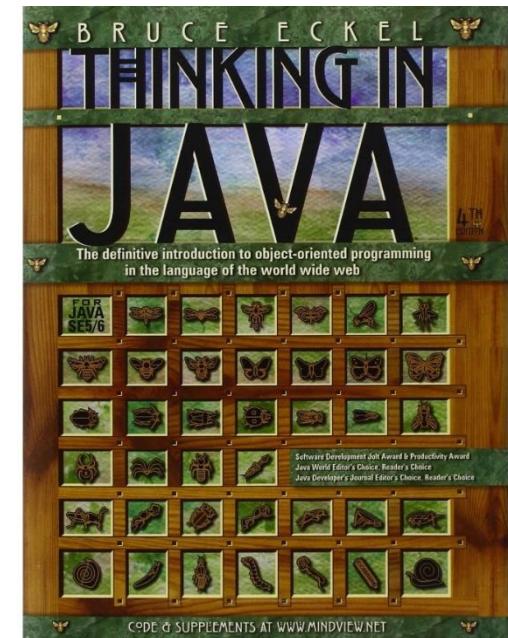
javIO.MyIOClass.java - CSC3100/src

44



More information

- ▶ Shortcut keys in IDE
 - E.g., in Eclipse IDE, to import packages automatically, we can use shortcut key:
“Shift” + “Ctrl” + “o”
- ▶ More online materials
 - <https://www.guru99.com/java-tutorial.html>
 - <https://fileadmin.cs.lth.se/cs/Education/EDA040/common/java21.pdf>
 - Book: Think in Java





Materials for self-learning



Example: Fibonacci Series

- ▶ In Fibonacci series, next number is the sum of previous two numbers.
 - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 etc.
 - The first two numbers of Fibonacci series are 0 and 1.
- ▶ Two ways to write the Fibonacci series program:
 - Fibonacci Series without using recursion
 - Fibonacci Series using recursion
- ▶ We will see
 - Defining variables
 - Loop
 - Recursion



```
1 class FibonacciExample1
2 {
3     public static void main(String args[])
4     {
5         int n1 = 0, n2 = 1, n3, i, count = 10;
6         System.out.print(n1 + " " + n2); //printing 0 and 1
7
8         for(i = 2; i < count; ++i) //loop starts from 2 because 0 and 1 are already printed
9         {
10             n3 = n1 + n2;
11             System.out.print(" " + n3);
12             n1 = n2;
13             n2 = n3;
14         }
15     }
16 }
17 }
```



```
1 class FibonacciExample2
2 {
3     static int n1 = 0, n2 = 1, n3 = 0;
4     static void printFibonacci(int count)
5     {
6         if(count > 0)
7         {
8             n3 = n1 + n2;
9             n1 = n2;
10            n2 = n3;
11            System.out.print(" " + n3);
12            printFibonacci(count - 1);
13        }
14    }
15    public static void main(String args[])
16    {
17        int count = 10;
18        System.out.print(n1 + " " + n2); //printing 0 and 1
19        printFibonacci(count - 2); //n-2 because 2 numbers are already printed
20    }
21 }
```



Example: Greatest Common Diviser

- ▶ Given two numbers, find the greatest common divisor (GCD)
 - for example, GCD of 15 and 20 is 5; GCD of 7 and 5 is 1.
- ▶ Algorithm 1
 - iterate from the smaller number given to 1
- ▶ Algorithm 2 (Euclid's Algorithm)
 - if we subtract the smaller number from the larger number, the GCD doesn't change
 - when the smaller number exactly divides the larger number, the smaller number is the GCD of the two given numbers.



```
3 public class gcd1 {
4     static int gcdByBruteForce(int n1, int n2) {
5         int gcd = 1;
6         for (int i = 1; i <= n1 && i <= n2; i++) {
7             if (n1 % i == 0 && n2 % i == 0) {
8                 gcd = i;
9             }
10        }
11        return gcd;
12    }
13    public static void main(String[] args) {
14        // TODO Auto-generated method stub
15        int a = 15, b = 5;
16        int g = gcdByBruteForce(a, b);
17        System.out.println("GCD of " + a + " and " + b + " is: " + g);
18    }
19}
20 }
```



```
3 public class gcd2 {
4•     static int gcdByEuclidsAlgorithm(int n1, int n2) {
5         if (n2 == 0) {
6             return n1;
7         }
8         return gcdByEuclidsAlgorithm(n2, n1 % n2);
9     }
10•    public static void main(String[] args) {
11        // TODO Auto-generated method stub
12        int a = 15, b = 5;
13        int g = gcdByEuclidsAlgorithm(a, b);
14        System.out.println("GCD of " + a + " and " + b + " is: " + g);
15    }
16
17 }
```



Example: Prime Numbers

- ▶ Verify prime number in Java: Prime number is a number that is greater than 1 and divided by 1 or itself only. For example, 2, 3, 5, 7, 11, 13, 17... are the prime numbers.



```
1 public class PrimeExample
2 {
3     public static void main(String args[])
4     {
5         int i, m = 0, flag = 0;
6         int n = 3; //it is the number to be checked
7         m = n / 2;
8         if(n == 0 || n == 1)
9         {
10             System.out.println(n + " is not prime number");
11         }
12         else
13         {
14             for(i = 2; i <= m; i++)
15             {
16                 if(n % i == 0)
17                 {
18                     System.out.println(n + " is not prime number");
19                     flag = 1;
20                     break;
21                 }
22             }
23             if(flag == 0)
24             {
25                 System.out.println(n + " is prime number");
26             }
27         }//end of else
28     }
29 }
```



Example: Palindrome

- ▶ A palindrome number is a number that is same after reverse.
 - 545, 151, 34543, 343, 171, 48984 are the palindrome numbers.
- ▶ Algorithm
 - Get the number to check for palindrome
 - Hold the number in temporary variable
 - Reverse the number
 - Compare the temporary number with reversed number
 - If both numbers are same, print "palindrome number"
 - Else print "not palindrome number"



```
1 class PalindromeExample
2 {
3     public static void main(String args[])
4     {
5         int r, sum = 0, temp;
6         int n = 454; //It is the number variable to be checked for palindrome
7
8         temp = n;
9         while(n > 0)
10        {
11            r = n % 10; //getting remainder
12            sum = (sum * 10) + r;
13            n = n / 10;
14        }
15        if(temp == sum)
16            System.out.println("palindrome number ");
17        else
18            System.out.println("not palindrome");
19    }
20 }
```

```
palindrome number
```



```
1 import java.util.*;
2 class PalindromeExample2
3 {
4     public static void main(String args[])
5     {
6         String original, reverse = ""; // Objects of String class
7         Scanner in = new Scanner(System.in);
8         System.out.println("Enter a string/number to check if it is a palindrome");
9         original = in.nextLine();
10        int length = original.length();
11        for ( int i = length - 1; i >= 0; i-- )
12            reverse = reverse + original.charAt(i);
13        if (original.equals(reverse))
14            System.out.println("Entered string/number is a palindrome.");
15        else
16            System.out.println("Entered string/number isn't a palindrome.");
17    }
18 }
```



Example: Object

- ▶ The object is a basic building block of an OOPs language. We cannot execute any Java program without creating an object.
- ▶ Five ways to create an object in Java.
 - Using new Keyword: the most popular way to create an object or instance of the class. When we create an instance of the class by using the new keyword, it allocates memory for the newly created **object** and also returns the **reference** of that object to that memory. The new keyword is also used to create an array.
 - Using clone() method
 - Using newInstance() method of the Class class
 - Using newInstance() method of the Constructor class
 - Using Deserialization



Object vs. Class

No.	Object	Class
1)	Object is an instance of a class.	Class is a blueprint or template from which objects are created.
2)	Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a group of similar objects .
3)	Object is a physical entity.	Class is a logical entity.
4)	Object is created through new keyword mainly e.g. Student s1=new Student();	Class is declared using class keyword e.g. class Student{}
5)	Object is created many times as per requirement.	Class is declared once .
6)	Object allocates memory when it is created .	Class doesn't allocated memory when it is created .
7)	There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only one way to define class in java using class keyword.



```
1 public class CreateObjectExample1
2 {
3     void show()
4     {
5         System.out.println("Welcome to javaTpoint");
6     }
7     public static void main(String[] args)
8     {
9         //creating an object using new keyword
10        CreateObjectExample1 obj = new CreateObjectExample1();
11        //invoking method using the object
12        obj.show();
13    }
14 }
```



Example: Printing ASCII value

- ▶ ASCII: American Standard Code for Information Interchange.
 - A 7-bit character set contains 128 (0 to 127) characters.
 - It represents the numerical value of a character.
 - Example: ASCII value of A is 65

- ▶ Two ways to print ASCII value:
 - Assigning a Variable to the int Variable
 - Using Type-Casting



```
1 public class PrintAsciiValueExample1
2 {
3     public static void main(String[] args)
4     {
5         // character whose ASCII value to be found
6         char ch1 = 'a';
7         char ch2 = 'b';
8         // variable that stores the integer value of the character
9         int asciivalue1 = ch1;
10        int asciivalue2 = ch2;
11        System.out.println("The ASCII value of " + ch1 + " is: " + asciivalue1);
12        System.out.println("The ASCII value of " + ch2 + " is: " + asciivalue2);
13    }
14 }
```

```
The ASCII value of a is: 97
The ASCII value of b is: 98
```



```
1 public class PrintAsciiValueExample2
2 {
3     public static void main(String[] String)
4     {
5         int ch1 = 'a';
6         int ch2 = 'b';
7         System.out.println("The ASCII value of a is: " + ch1);
8         System.out.println("The ASCII value of b is: " + ch2);
9     }
10 }
```

```
The ASCII value of a is: 97
The ASCII value of b is: 98
```



Example: Count Characters

- ▶ Iterate through the string and count the characters.
 - STEP 1: START
 - STEP 2: DEFINE String *string* = "The best of both worlds".
 - STEP 3: SET *count* = 0.
 - STEP 4: SET *i*=0. REPEAT STEP 5 to STEP 6 as long as *i*<*string.length*
 - STEP 5: IF (*string.charAt(i)*!= ' ') then *count* = *count* + 1.
 - STEP 6: *i*=*i*+1
 - STEP 7: PRINT *count*.
 - STEP 8: END



```
1 public class CountCharacter
2 {
3     public static void main(String[] args)
4     {
5         String string = "The best of both worlds";
6         int count = 0;
7
8         //Counts each character except space
9         for(int i = 0; i < string.length(); i++)
10        {
11            if(string.charAt(i) != ' ')
12                count++;
13        }
14
15        //Displays the total number of characters present in the given string
16        System.out.println("Total number of characters in a string: " + count);
17    }
18 }
```

```
Total number of characters in a string: 19
```



Example: Count Punctuation Characters

▶ Algorithm

- Define a string or read from the user.
- Declare a variable to count the number of punctuations and initialized it with 0.
- Match each character with the punctuation marks (!, ., ', -, ", ?, ;). If any character in the string is matched, increase the count variable by 1.
- Print the count variable that gives the total number of punctuations.



```
1 public class CountPunctuation
2 {
3     public static void main (String args[])
4     {
5         //Stores the count of punctuation marks
6         int count = 0;
7         String str = "He said, 'The mailman loves you.' I heard it with my own ears.";
8         for (int i = 0; i < str.length(); i++)
9         {
10             //Checks whether given character is punctuation mark
11             if(str.charAt(i) == '!' || str.charAt(i) == ',' || str.charAt(i) == ';' || str.
12                 charAt(i) == '.' || str.charAt(i) == '?' || str.charAt(i) == '-' ||
13                 str.charAt(i) == '\'' || str.charAt(i) == '\"' || str.charAt(i) == ':')
14             {
15                 count++;
16             }
17         }
18     }
19 }
```

The number of punctuations exists in the string is: 5



Example: Replace Characters

- ▶ Replace lower-case characters in a string to upper-case and vice versa.
 - STEP 1: START
 - STEP 2: DEFINE a string $str = "Great\ Power"$.
 - STEP 3: DEFINE $newstr$ as StringBuffer object .
 - STEP 4: SET $i=0$. REPEAT STEP 5 to STEP 6 as long as $i < str.length()$.
 - STEP 5: IF lower-case character encountered then CONVERT in upper-case. ELSEIF upper-case character encountered then CONVERT in lower-case.
 - STEP 6: $i=i+1$
 - STEP 7: PRINT $newstr$.
 - STEP 8: END

```
1 public class changeCase
2 {
3     public static void main(String[] args)
4     {
5
6         String str1 = "Great Power";
7         StringBuffer newStr = new StringBuffer(str1);
8
9         for(int i = 0; i < str1.length(); i++)
10        {
11
12             //Checks for lower case character
13             if(Character.isLowerCase(str1.charAt(i)))
14             {
15                 //Convert it into upper case using toUpperCase() function
16                 newStr.setCharAt(i, Character.toUpperCase(str1.charAt(i)));
17             }
18             //Checks for upper case character
19             else if(Character.isUpperCase(str1.charAt(i)))
20             {
21                 //Convert it into upper case using toLowerCase() function
22                 newStr.setCharAt(i, Character.toLowerCase(str1.charAt(i)));
23             }
24         }
25         System.out.println("String after case conversion : " + newStr);
26     }
27 }
```

```
String after case conversion: gREAT pOWER
```



Example: check rotation of a string

- ▶ Check whether string 2 is a rotation of string 1:
Concatenate string 1 with string 1. If string 2 is present in concatenated string then, string 2 is rotation of string 1.
 - STEP 1: START
 - STEP 2: DEFINE String $str1 = "abcde"$, $str2 = "deabc"$
 - STEP 3: IF length of $str1$ not equals to $str2$ then PRINT "No" else go to STEP 4
 - STEP 4: CONCATENATE $str1$ with $str1$.
 - STEP 5: IF $str2$ present in $str1$ then PRINT "Yes" else PRINT "No".
 - STEP 6: END



```
1 public class StringRotation
2 {
3     public static void main(String[] args)
4     {
5         String str1 = "abcde", str2 = "deabc";
6
7         if(str1.length() != str2.length())
8         {
9             System.out.println("Second string is not a rotation of first string");
10        }
11        else
12        {
13            //Concatenate str1 with str1 and store it in str1
14            str1 = str1.concat(str1);
15            //Check whether str2 is present in str1
16            if(str1.indexOf(str2) != -1)
17                System.out.println("Second string is a rotation of first string");
18            else
19                System.out.println("Second string is not a rotation of first string");
20        }
21    }
22 }
```

Second string is a rotation of first string



Example: file handling (I/O)

- ▶ The File class is an abstract representation of file and directory pathname.
 - A pathname can be either absolute or relative.
- ▶ The File class have methods for working with directories and files:
 - creating new directories or files
 - deleting and renaming directories or files
 - listing the contents of a directory, etc.
- ▶ **VERY USEFUL IN THIS CLASS**

Modifier and Type	Method	Description
static File	createTempFile(String prefix, String suffix)	It creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
boolean	createNewFile()	It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
boolean	canWrite()	It tests whether the application can modify the file denoted by this abstract pathname.
boolean	canExecute()	It tests whether the application can execute the file denoted by this abstract pathname.
boolean	canRead()	It tests whether the application can read the file denoted by this abstract pathname.
boolean	isAbsolute()	It tests whether this abstract pathname is absolute.
boolean	isDirectory()	It tests whether the file denoted by this abstract pathname is a directory.
boolean	isFile()	It tests whether the file denoted by this abstract pathname is a normal file.
String	getName()	It returns the name of the file or directory denoted by this abstract pathname.
String	getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
Path	toPath()	It returns a java.nio.file.Path object constructed from the this abstract path.
URI	toURI()	It constructs a file: URI that represents this abstract pathname.
File[]	listFiles()	It returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname
long	getFreeSpace()	It returns the number of unallocated bytes in the partition named by this abstract path name.
String[]	list(FilenameFilter filter)	It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
boolean	mkdir()	It creates the directory named by this abstract pathname.

```
1 import java.io.*;
2 public class FileDemo
3 {
4     public static void main(String[] args)
5     {
6
7         try
8         {
9             File file = new File("javaFile123.txt");
10            if (file.createNewFile())
11            {
12                System.out.println("New File is created!");
13            }
14            else
15            {
16                System.out.println("File already exists.");
17            }
18        }
19        catch (IOException e)
20        {
21            e.printStackTrace();
22        }
23
24    }
25 }
```

New File is created!

```
1 import java.io.*;
2 public class FileDemo2
3 {
4     public static void main(String[] args)
5     {
6
7         String path = "";
8         boolean bool = false;
9         try
10        {
11            // createing new files
12            File file = new File("testFile1.txt");
13            file.createNewFile();
14            System.out.println(file);
15            // createing new canonical from file object
16            File file2 = file.getCanonicalFile();
17            // returns true if the file exists
18            System.out.println(file2);
19            bool = file2.exists();
20            // returns absolute pathname
21            path = file2.getAbsolutePath();
22            System.out.println(bool);
23            // if file exists
24            if (bool)
25            {
26                // prints
27                System.out.print(path + " Exists? " + bool);
28            }
29        }
30        catch (Exception e)
31        {
32            // if any error occurs
33            e.printStackTrace();
34        }
35    }
36 }
```

```
testFile1.txt
/home/Work/Project/File/testFile1.txt
true
/home/Work/Project/File/testFile1.txt Exists? true
```

```
1 import java.io.*;
2 public class FileExample
3 {
4     public static void main(String[] args)
5     {
6         File f = new File("/Users/sonoojaiswal/Documents");
7         String filenames[] = f.list();
8         for(String filename : filenames)
9         {
10             System.out.println(filename);
11         }
12     }
13 }
```

```
"info.properties"
"info.properties".rtf
.DS_Store
.localized
Alok news
apache-tomcat-9.0.0.M19
apache-tomcat-9.0.0.M19.tar
bestreturn_org.rtf
BIODATA.pages
BIODATA.pdf
BIODATA.png
struts2jars.zip
workspace
```

```
1 import java.io.*;
2 public class FileExample
3 {
4     public static void main(String[] args)
5     {
6         File dir = new File("/Users/Documents");
7         File files[] = dir.listFiles();
8         for(File file : files)
9         {
10             System.out.println(file.getName() + " Can Write: " + file.canWrite() + "Is Hidden: "
11                         + file.isHidden() + " Length: " + file.length() + " bytes");
12         }
13     }
}
```

```
"info.properties" Can Write: true Is Hidden: false Length: 15 bytes
"info.properties".rtf Can Write: true Is Hidden: false Length: 385 bytes
.DS_Store Can Write: true Is Hidden: true Length: 36868 bytes
.localized Can Write: true Is Hidden: true Length: 0 bytes
Alok news Can Write: true Is Hidden: false Length: 850 bytes
apache-tomcat-9.0.0.M19 Can Write: true Is Hidden: false Length: 476 bytes
apache-tomcat-9.0.0.M19.tar Can Write: true Is Hidden: false Length: 13711360 bytes
bestreturn_org.rtf Can Write: true Is Hidden: false Length: 389 bytes
BIODATA.pages Can Write: true Is Hidden: false Length: 707985 bytes
BIODATA.pdf Can Write: true Is Hidden: false Length: 69681 bytes
BIODATA.png Can Write: true Is Hidden: false Length: 282125 bytes
workspace Can Write: true Is Hidden: false Length: 1972 bytes
```



Example: fileinputstream & fileoutputstream

- ▶ Java FileInputStream class obtains input bytes from a file.
 - used for reading byte-oriented data (streams of raw bytes) such as image, audio etc.
 - can also read character-stream data.
 - For reading streams of characters, it is preferred to use FileReader.

- ▶ FileOutputStream write primitives values into a file.
 - Write byte-oriented and character-oriented data.
 - For character-oriented data, it is preferred to use FileWriter.

<https://www.javatpoint.com/java-fileinputstream-class>

<https://www.javatpoint.com/java-fileoutputstream-class>



Java FileInputStream class methods

Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to b.length bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to len bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream .

```
1 import java.io.FileInputStream;
2 public class DataStreamExample
3 {
4     public static void main(String args[])
5     {
6         try
7         {
8             FileInputStream fin = new FileInputStream("D:\\testout.txt");
9             int i = fin.read();
10            System.out.print((char)i);
11
12            fin.close();
13        }
14        catch(Exception e)
15        {
16            System.out.println(e);
17        }
18    }
19 }
```

Note: Before running the code, a text file named as "testout.txt" is required to be created. In this file, we are having following content:

```
Welcome to javatpoint.
```

After executing the above program, you will get a single character from the file which is 87 (in byte form). To see the text, you need to convert it into character.

Output:

```
W
```



```
1 import java.io.FileInputStream;
2 public class DataStreamExample
3 {
4     public static void main(String args[])
5     {
6         try
7         {
8             FileInputStream fin = new FileInputStream("D:\\testout.txt");
9             int i = 0;
10            while((i = fin.read()) != -1)
11            {
12                System.out.print((char)i);
13            }
14            fin.close();
15        }
16        catch(Exception e)
17        {
18            System.out.println(e);
19        }
20    }
21 }
```



FileOutputStream class methods

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write ary.length bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write len bytes from the byte array starting at offset off to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.



```
1 import java.io.FileOutputStream;
2 public class FileOutputStreamExample
3 {
4     public static void main(String args[])
5     {
6         try
7         {
8             FileOutputStream fout = new FileOutputStream("D:\\\\testout.txt");
9             fout.write(65);
10            fout.close();
11            System.out.println("success...");
12        }
13        catch(Exception e)
14        {
15            System.out.println(e);
16        }
17    }
18 }
```

Success...

The content of a text file **testout.txt** is set with the data **A**.

testout.txt

A



```
1 import java.io.FileOutputStream;
2 public class FileOutputStreamExample
3 {
4     public static void main(String args[])
5     {
6         try
7         {
8             FileOutputStream fout = new FileOutputStream("D:\\testout.txt");
9             String s = "Welcome to javaTpoint.";
10            byte b[] = s.getBytes(); //converting string into byte array
11            fout.write(b);
12            fout.close();
13            System.out.println("success...");
14        }
15        catch(Exception e)
16        {
17            System.out.println(e);
18        }
19    }
20 }
```

Success...

The content of a text file **testout.txt** is set with the data **Welcome to javaTpoint**.

testout.txt

Welcome to javaTpoint.



Example: filereader & filewriter

- ▶ Both FileReader and FileWriter classes are character-oriented (for text file)
 - FileReader is used to read data from the file.
 - It returns data in byte format like FileInputStream class.
 - FileWriter is used to write character-oriented data to a file.
 - Unlike FileOutputStream class, no need to convert string into byte array.



```
1 import java.io.FileReader;
2 public class FileReaderExample
3 {
4     public static void main(String args[])throws Exception
5     {
6         FileReader fr = new FileReader("D:\\testout.txt");
7         int i;
8         while((i = fr.read()) != -1)
9             System.out.print((char)i);
10        fr.close();
11    }
12 }
```

Here, we are assuming that you have following data in "testout.txt" file:

```
Welcome to javaTpoint.
```

Output:

```
Welcome to javaTpoint.
```



```
1 import java.io.FileWriter;
2 public class FileWriterExample
3 {
4     public static void main(String args[])
5     {
6         try
7         {
8             FileWriter fw = new FileWriter("D:\\testout.txt");
9             fw.write("Welcome to javaTpoint.");
10            fw.close();
11        }
12        catch(Exception e)
13        {
14            System.out.println(e);
15        }
16        System.out.println("Success...");
17    }
18 }
```