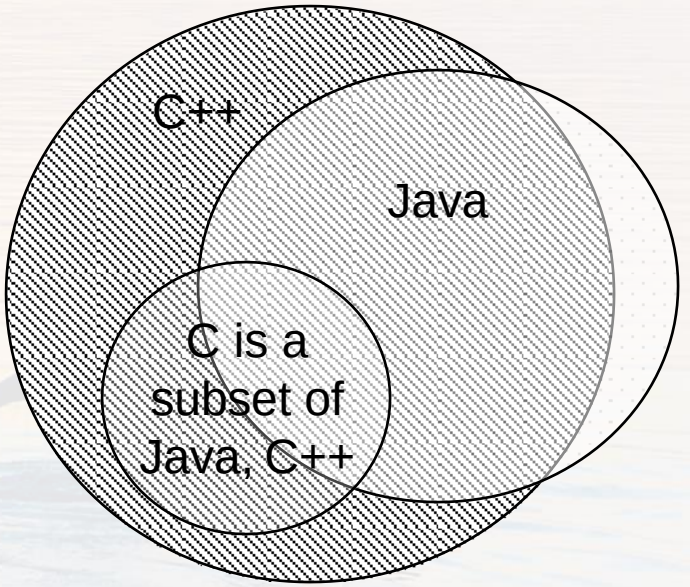
Java Programming

CHAPTER 1 & 2

Introduction to Objects

Contents

- The Java Technology Phenomenon
- The " Welcome to Java!" Application
- What is an Object?
- What is a Class?
- Online Documentation



The Java Technology Phenomenon

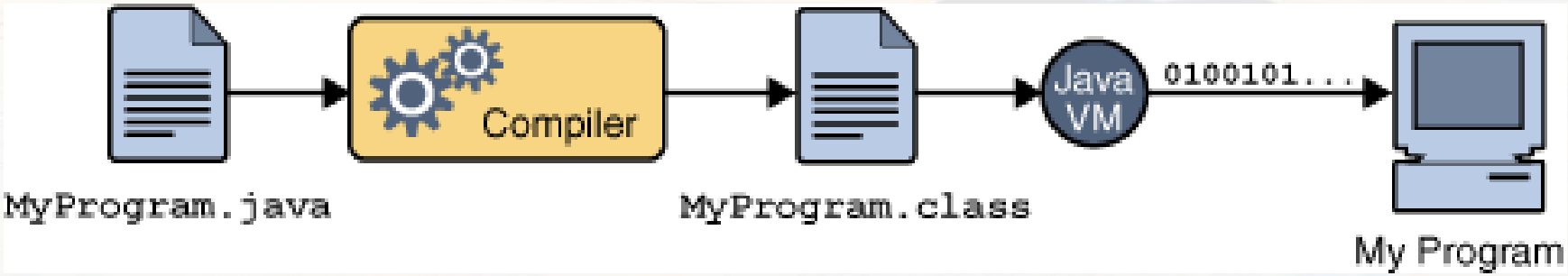
- Java technology
  - The Java programming language
  - The Java platform
- The Java programming language is a high-level language that can be characterized:

Simple	Architecture neutral
Object oriented	Portable
Distributed	High performance
Multithreaded	Robust
Dynamic	Secure

Each of these words is explained in *The Java Language Environment* http://www.oracle.com/technetwork/java/langenv-140151.html

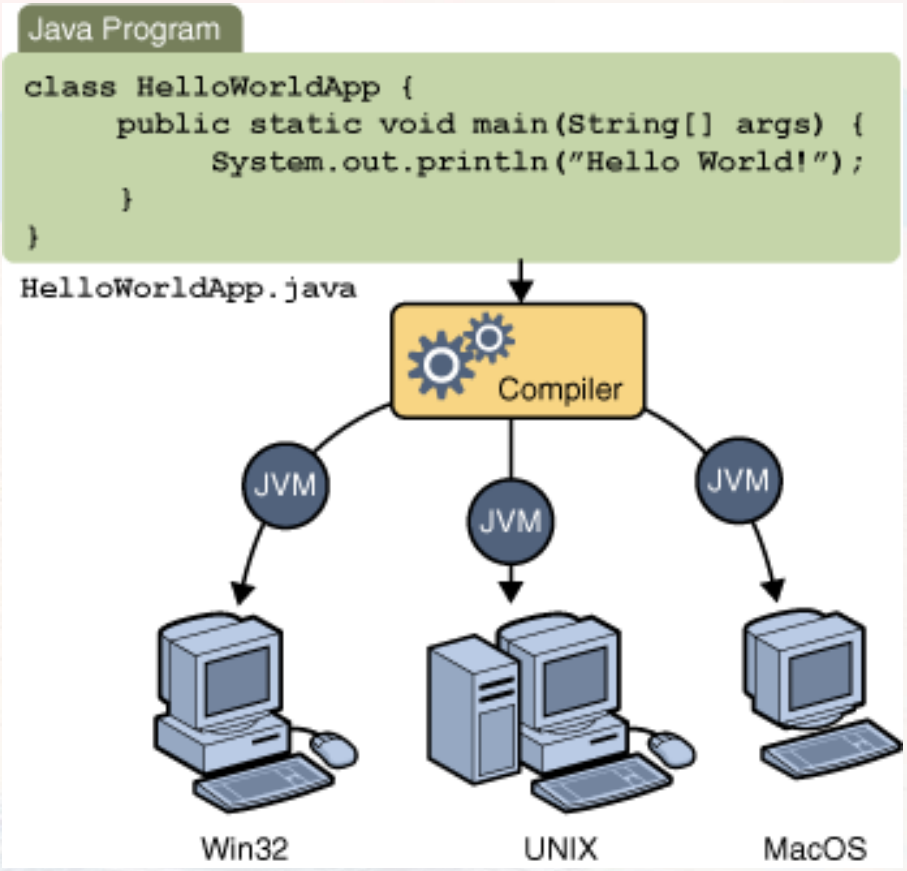
The Java programming language

- All source code is first written in plain text files ending with the *.java* extension.
- Those source files are then compiled into .class files by the *javac* compiler. A *.class* file does not contain code that is native to your processor; it contains *bytecodes* — the machine language of the Java Virtual Machine (Java VM).
- The java launcher tool then runs your application with an instance of the Java Virtual Machine.



The Java VM

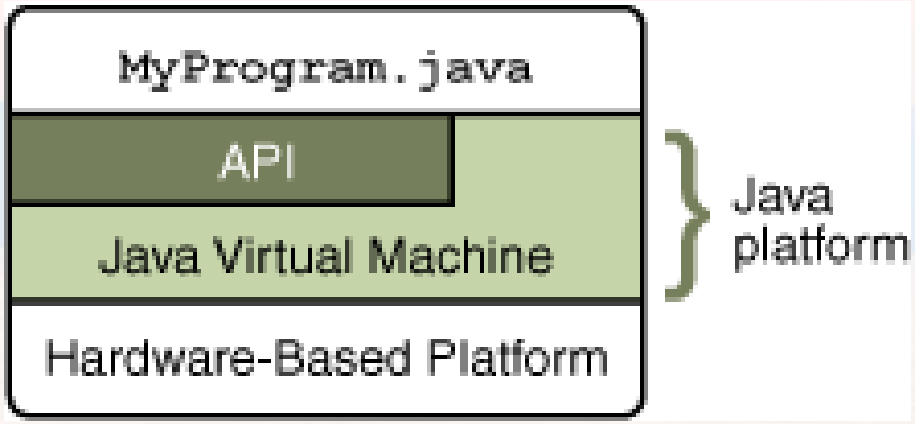
- The Java VM is available on many different operating systems.
- The same *.class* files may run on
  - Microsoft Windows,
  - the Solaris ™ Operating System (Solaris OS),
  - Linux, or
  - Mac OS.



The Java Platform

- The Java platform has two components:
  - The *Java Virtual Machine*
  - The *Java Application Programming Interface* (API)
- The API is a large collection of ready-made software components.
- The API provides many useful capabilities:
  - It is grouped into libraries of related classes and interfaces;
  - These libraries are known as *packages*.

The Java Platform



- The API and Java Virtual Machine insulate (protect) the program from the underlying hardware.
- As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without influencing portability.

What Can Java Technology Do?

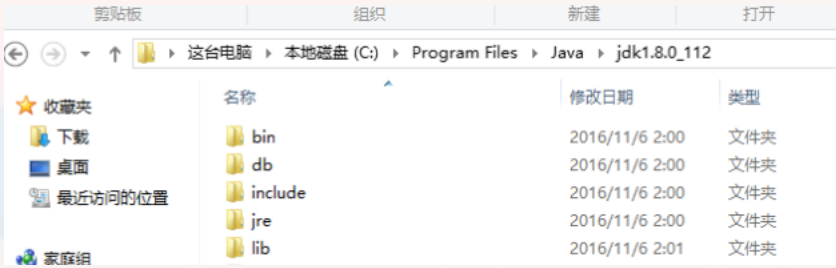
- **Development Tools**: They provide everything you need for compiling, running, monitoring, debugging, and documenting your applications. The main tools are
  - The compiler: *javac*
  - The launcher: *java*
  - the documentation tool: *javadoc*.
- **Application Programming Interface (API)**: The API provides the core functionality of the Java language:
  - It offers a wide array of useful classes ready for use in your own applications.
  - It spans everything from basic objects, to networking and security, to XML generation and database access, and more.
  - The core API is very large: The Java SE Development Kit 6 (JDK™ 6) documentation http://docs.oracle.com/javase/6/docs/index.html.

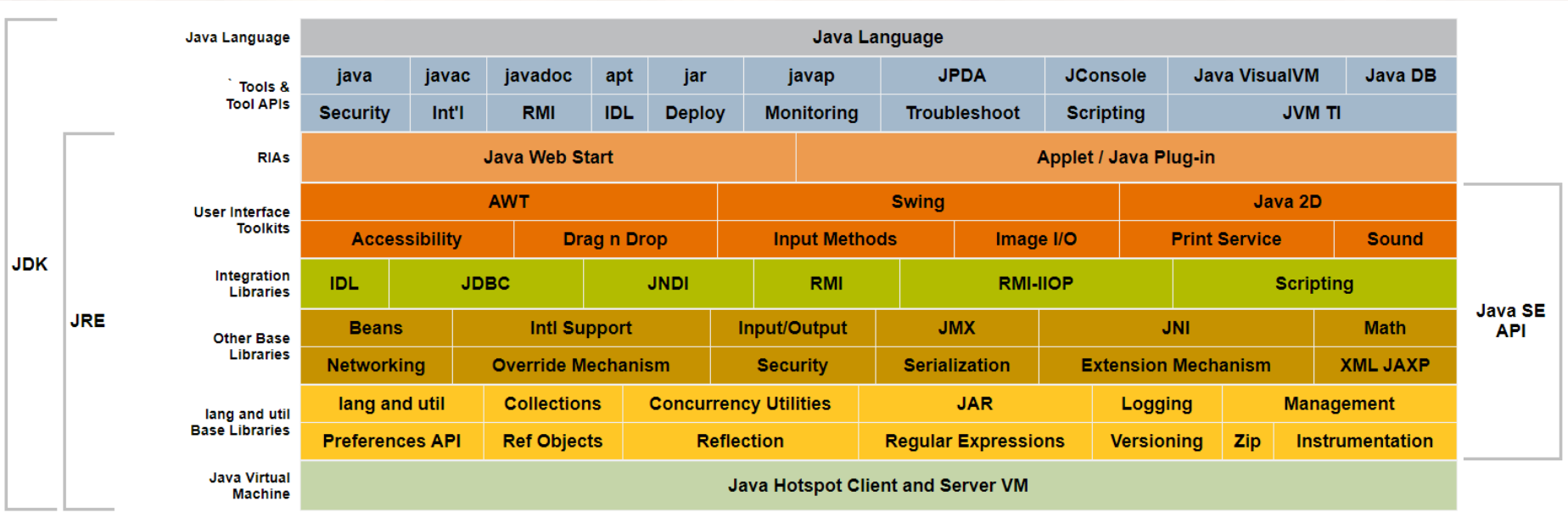
What Can Java Technology Do?

- **Deployment Technologies**: The JDK software provides standard mechanisms such as the Java Web Start software and Java Plug-In software for deploying your applications to end users.
- **User Interface Toolkits**: The Swing and Java 2D toolkits make it possible to create sophisticated Graphical User Interfaces (GUIs).
- **Integration Libraries**: Integration libraries such as the Java IDL API, JDBC™ API, Java Naming and Directory Interface™ ("J.N.D.I.") API, Java RMI, and Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology) enable database access and manipulation of remote objects.

What Can Java Technology Do?

- The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language.
- The Java Development Kit (JDK) is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications.





The " Welcome to Java!" Application

```
/**
 * The Welcome class
 * implements an application that
 * simply displays "Welcome to Java!"
 * to the standard output.
 */
public class Welcome {
    public static void main(String[] args) {
    System.out.println("Welcome to Java!");
 //Display the string.
     }
}
```

- How to create, compile and run:
  - Start your favorite text editor (e.g. Vim, emacs).
  - Type the code (left side of the slide).
  - Save the text to the file: Welcome.java
  - Compile the program typing: javac Welcome.java
  - Run the program typing: java Welcome
  - Output of the program: Welcome to Java!

A Closer Look at the "Welcome to Java!" Application

- The "Welcome to Java" application consists of three primary components:
  - source code comments,
  - the Welcome class definition, and
  - the main method.

- Try to find out those keys:
  - Class name
  - Statements
  - Reserved words
  - Main method
  - Statement terminator

Class name

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter. In this example, the class name is Welcome.

Main Method

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Line 2 defines the main method. In Java, every application must contain a main method.

In order to run a class, the class must contain a method named main. The program is executed from the main method.

Statement

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

A statement represents an action or a sequence of actions. The statement `System.out.println("Welcome to Java!")` in the program in Listing 1.1 is a statement to display the greeting "Welcome to Java!".

Statement Terminator

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Every statement in Java ends with a semicolon (;).

Reserved words

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word class, it understands that the word after class is the name for the class.

Get Started #1

Step 1: Create and edit “HelloWorld.java”

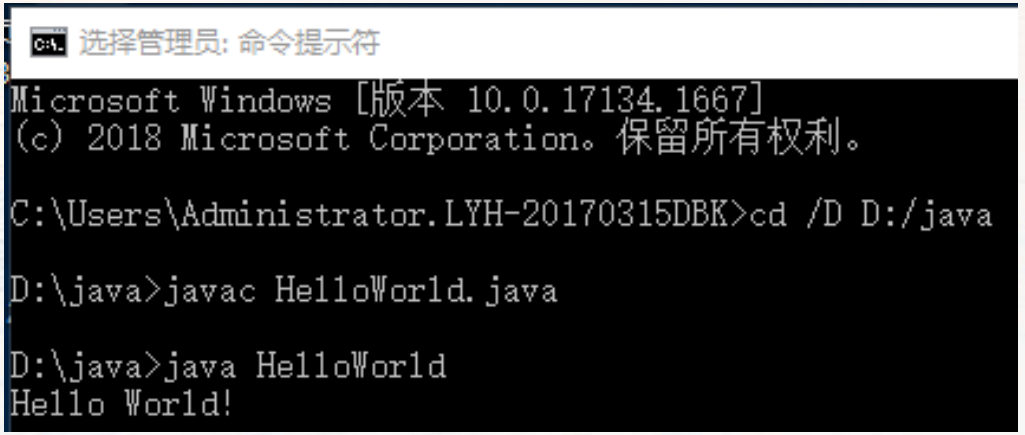
```
public class HelloWorld{
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Step 2: Compile “HelloWorld.java” to “HelloWorld.class”

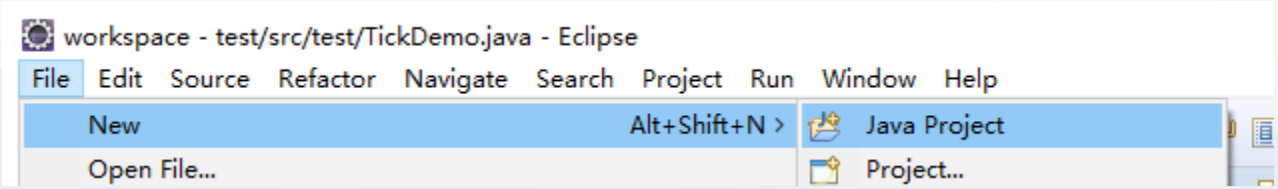
```
javac HelloWorld.java
```

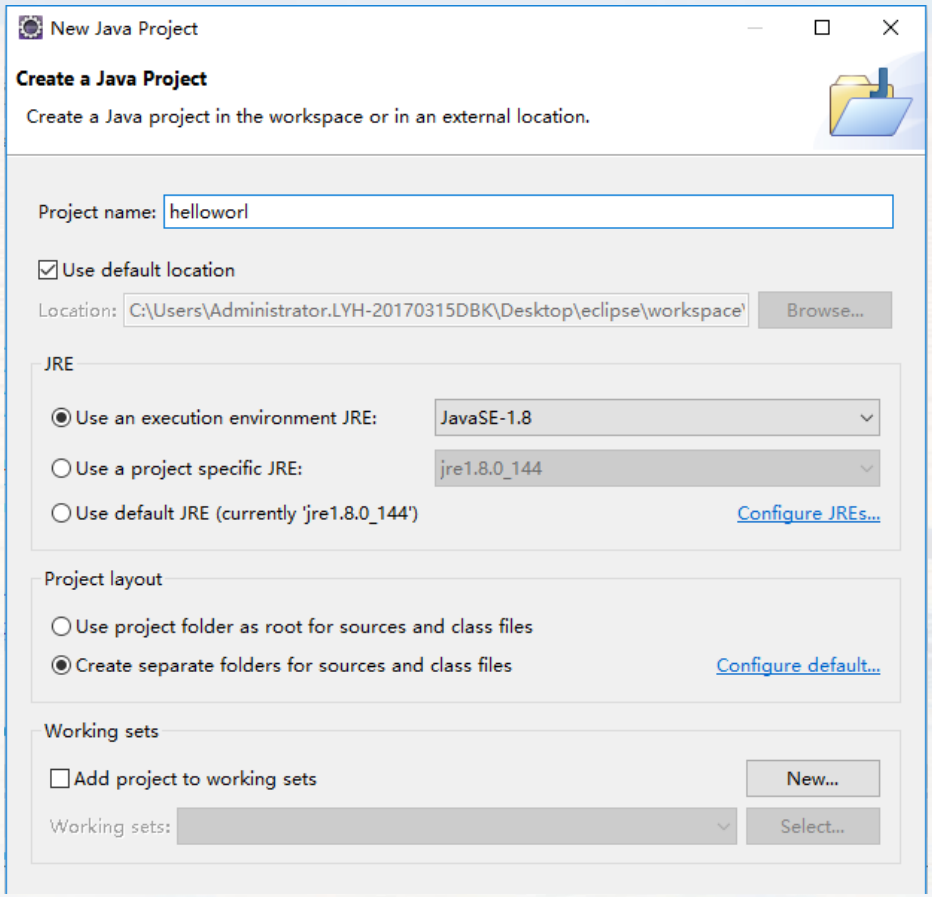
Step 3: Run java programing

```
java HelloWorld
```

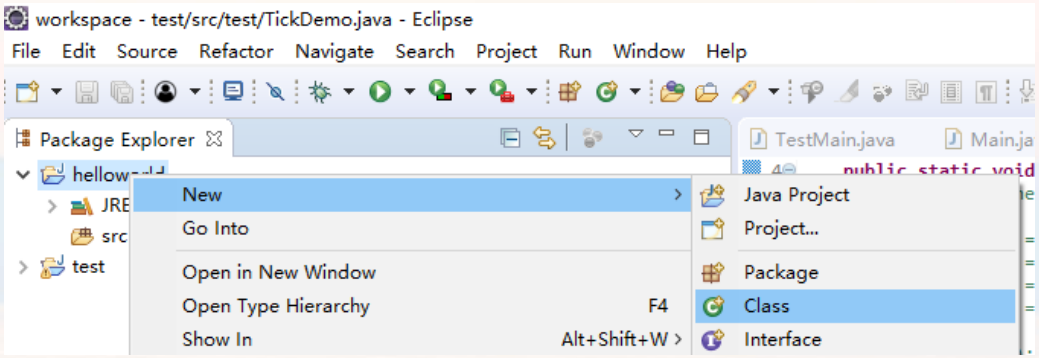


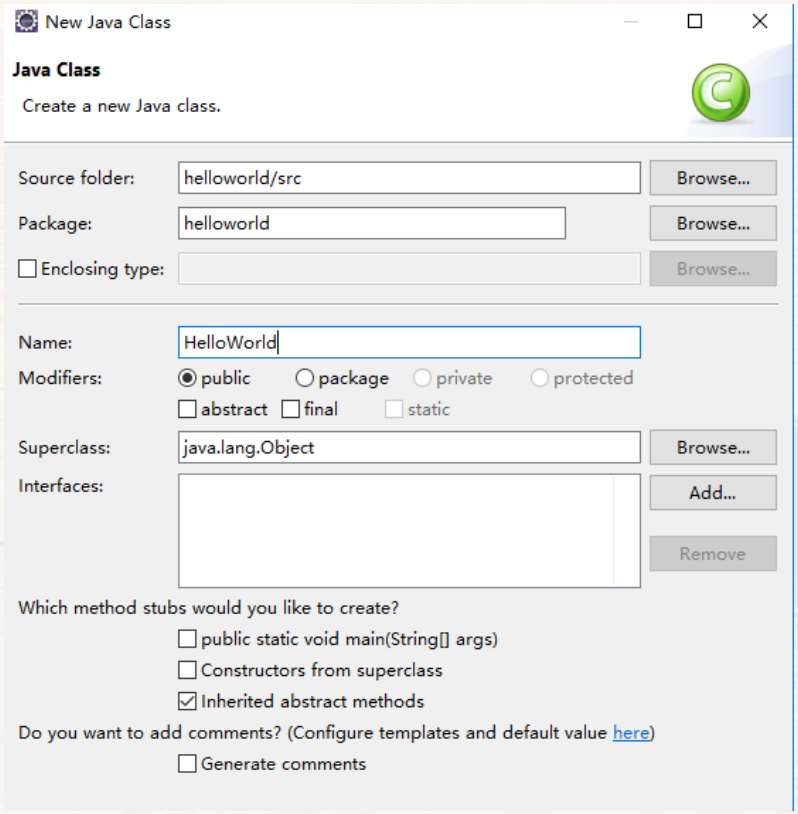
Get Started #2 – New Project



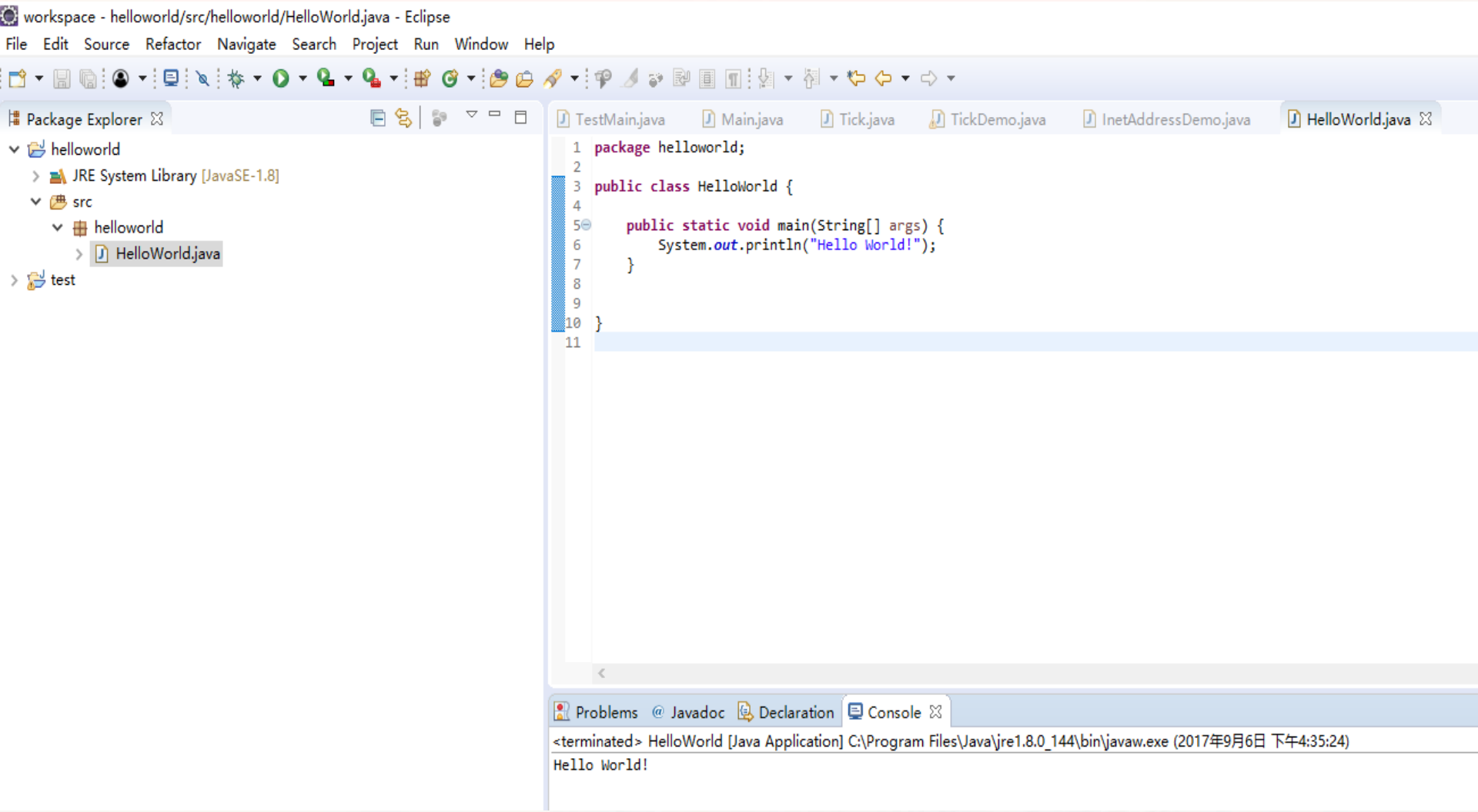


Get Started #2 – New Class





Get Started #2 - Hello World



Development Environment Installation

JDK Download:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

JDK configuration:

https://docs.oracle.com/javase/tutorial/essential/environment/paths.html

http://jingyan.baidu.com/article/363872ecd62f5f6e4ba16fcb.html

Eclipse Download:

http://www.eclipse.org/downloads/packages/release/indigo/sr2/eclipse-ide-java-developers

What is an Object?

- Objects are key to understanding object-oriented technology.
- Real world objects are around us (e.g., dogs, bicycles, cars, houses, tables, people).
- Objects share 2 characteristics:
  - State – the data of interest (e.g., people have a name, hair color, date of birth, etc.)
  - Behavior – what objects do (e.g., people walk, eat, read, etc.)

Example

- Example: Dogs have
  - state (or properties)
    - —name, color, eye color, height, length, weight.
  - behavior (or methods)
    - —barking, fetching, sitting, laying down, wagging tail.

Example – A Car

Color:White

Name:RAV4

State

Behavior

Wheel

Stop

Wheels:4

Height:1.65m

Weight:1.7 t

…

Straightforward

Move backward

You can add more states and behaviors, such as: owner, life_time……

Software Objects

- Software objects are conceptually similar to real-world objects: They consist of state and related behavior.
  - An object stores its state in *fields* (variables in some programming languages).
  - An object exposes its behavior through *methods* (functions in some programming languages). Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.

What is Encapsulation?

- Hiding internal state and requiring all interaction to be performed through an object's methods is known as *data encapsulation* — a fundamental principle of object-oriented programming.
  - In other words, there is not direct access to the fields of the object.

Software Objects: Benefits

- Modularity
  - The source code for an object can be written and maintained independently of the source code for other objects.
- Information-hiding
  - By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
- Code re-use
  - If an object already exists (perhaps written by another software developer), you can use that object in your program.
- Pluggability and debugging ease
  - If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace *it*, not the entire machine.

What Is a Class?

- In the real world, there are many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model.
- Each bicycle was built from the same set of blueprints (detailed plan, pattern) and therefore contains the same components.
- In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles.

Class Bicycle

Two objects of the Bicycle class

What Is a Class?

- A *class* is the blueprint (detailed plan, template) from which individual objects are created.
- It defines the characteristics and behaviors of all objects of a certain type.
- In other words, a class is an abstraction that contains the attributes and behaviors common to all objects of a given type.

Implementation of a Bicycle

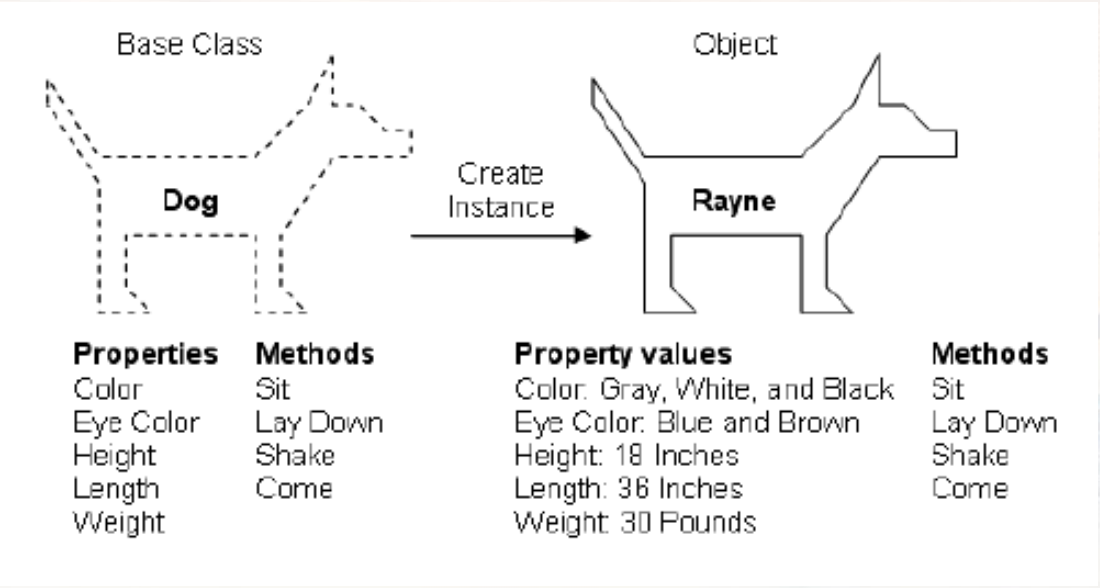
```
// File: BicycleDemo.java
class Bicycle {          // fields or attributes
    int cadence = 0;
    int speed = 0;
    int gear = 1;
    Bicycle()  {         // constructor
        cadence = 0;
        speed = 0;
        gear = 1;
    }                    // methods
    void changeCadence(int newValue) {
        cadence = newValue;
    }
    void changeGear(int newValue) {
        gear = newValue;
    }
    void speedUp(int increment) {
        speed = speed + increment;
    }
    void applyBrakes(int decrement) {
        speed = speed - decrement;
    }
    void printStates() { System.out.println( "cadence:"
    +  cadence+ " speed:"+speed+" gear:"+gear);
    }
} // end of the Bicycle class

class BicycleDemo {
    public static void main(String[] args) {
        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();
        bike1.speedUp(10);  // Invoke methods on those
        bike1.printStates();    // objects
        bike2.changeCadence(50);
        bike2.speedUp(15);
        bike2.printStates();
    }
}
```

Comments on the Previous Slide

- The fields cadence, speed, and gear represent the object's state, and the methods (changeCadence, changeGear, speedUp etc.) define its interaction with the outside world.
- The responsibility of creating and using new Bicycle objects belongs to the BicycleDemo class.
- Compile
  - javac BicycleDemo.java
- Run:
  - java BicycleDemo
- Output?

What is the Difference between a Class and a Object?

- One class – many objects: The class tells the Java virtual machine how to make an object of that particular type. Each object made from that class can have its own values for the instance variables.



Procedural versus Object-Oriented

DATA

```
typedef struct people {          // type
    char* name;
    char hairColor[32];
    char dateOfBirth[128];

} People;
```

PROCEDURES

```
People* createPeople(char* name, …)


void eat(People* people)
void read(People* people)
void walk(People* people)
```

DATA & PROCEDURES: *encapsulation*

```
class People {          // type
    private String name;
    private String hairColor;
    private String dateOfBirth;


                            // constructor
    People(String name,String color,…)

                            // methods
    public void eat()
    public void read()
    public walk()

}
```

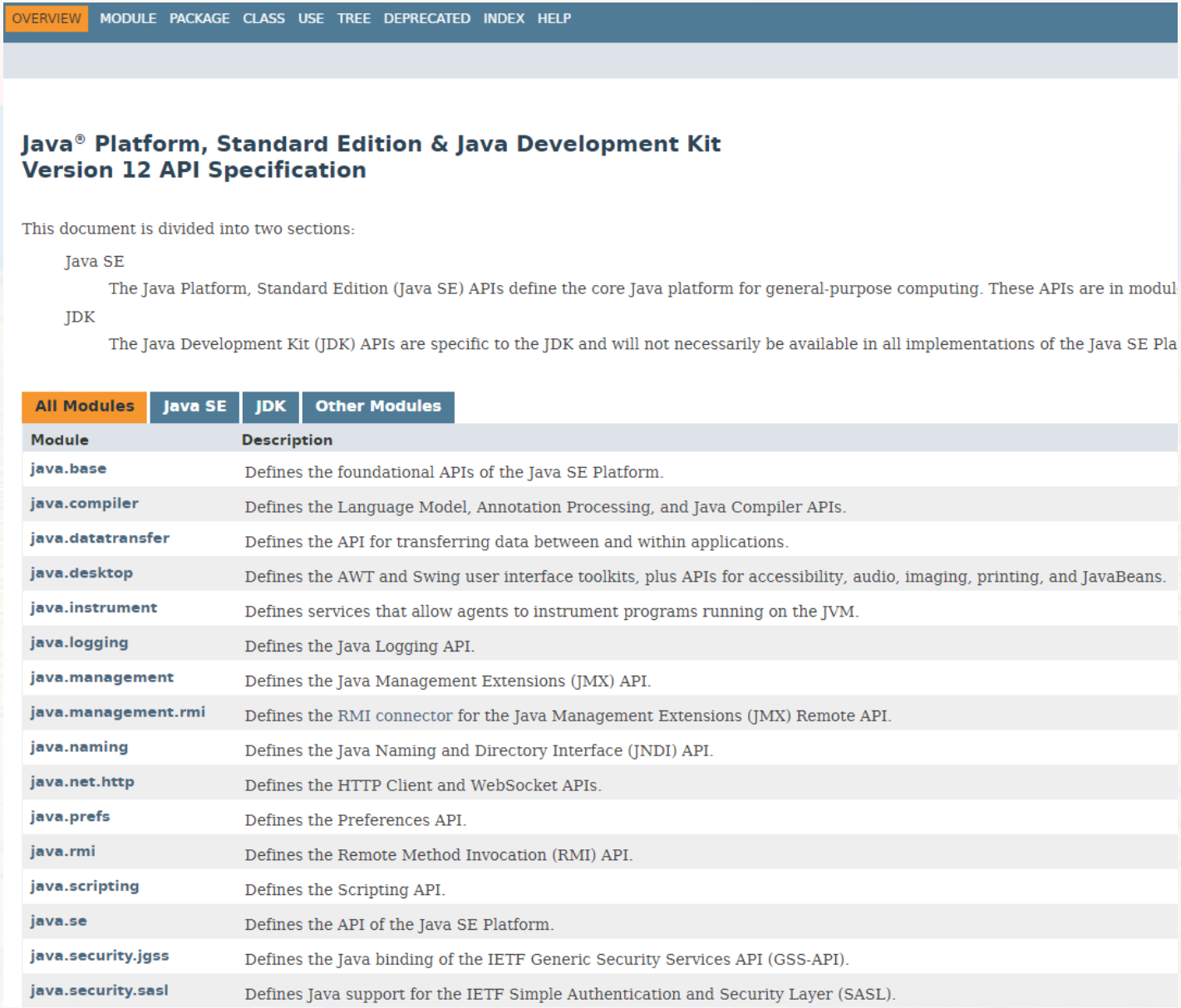
Comments on the Previous Slide

- Procedural approach:
  - Functions are introduced to reduce the size of the programs, improve readability in them, and simplify the debugging process of large programs.
  - The original data may easily get corrupted:
    - The data are accessible to all the functions, even to those which do not have any right to access them.
- Object-oriented approach:
  - The data and the functions, which are supposed to have the access to the data, are put into one box known as an object.
  - There are no chances of any unauthorized access to the data.
  - See slide: Software Objects: Benefits (Sl. 28).

Online Documentation

- Java provides online documentation for the whole environment:
  - How to compile and execute programs;
  - JDK classes and their methods;
  - Many example programs;
  - Many documents that address different topics in Java.
- http://docs.oracle.com/javase/12/

Java API Documentation



OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java® Platform, Standard Edition & Java Development Kit Version 12 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modul

JDK

The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Pla

All Modules | Java SE | JDK | Other Modules

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.
java.logging	Defines the Java Logging API.
java.management	Defines the Java Management Extensions (JMX) API.
java.management.rmi	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
java.naming	Defines the Java Naming and Directory Interface (JNDI) API.
java.net.http	Defines the HTTP Client and WebSocket APIs.
java.prefs	Defines the Preferences API.
java.rmi	Defines the Remote Method Invocation (RMI) API.
java.scripting	Defines the Scripting API.
java.se	Defines the API of the Java SE Platform.
java.security.jgss	Defines the Java binding of the IETF Generic Security Services API (GSS-API).
java.security.sasl	Defines Java support for the IETF Simple Authentication and Security Layer (SASL).