61FIT3JSD - Java Software Development

Lecture 1

Java Language Review

Lecture outline

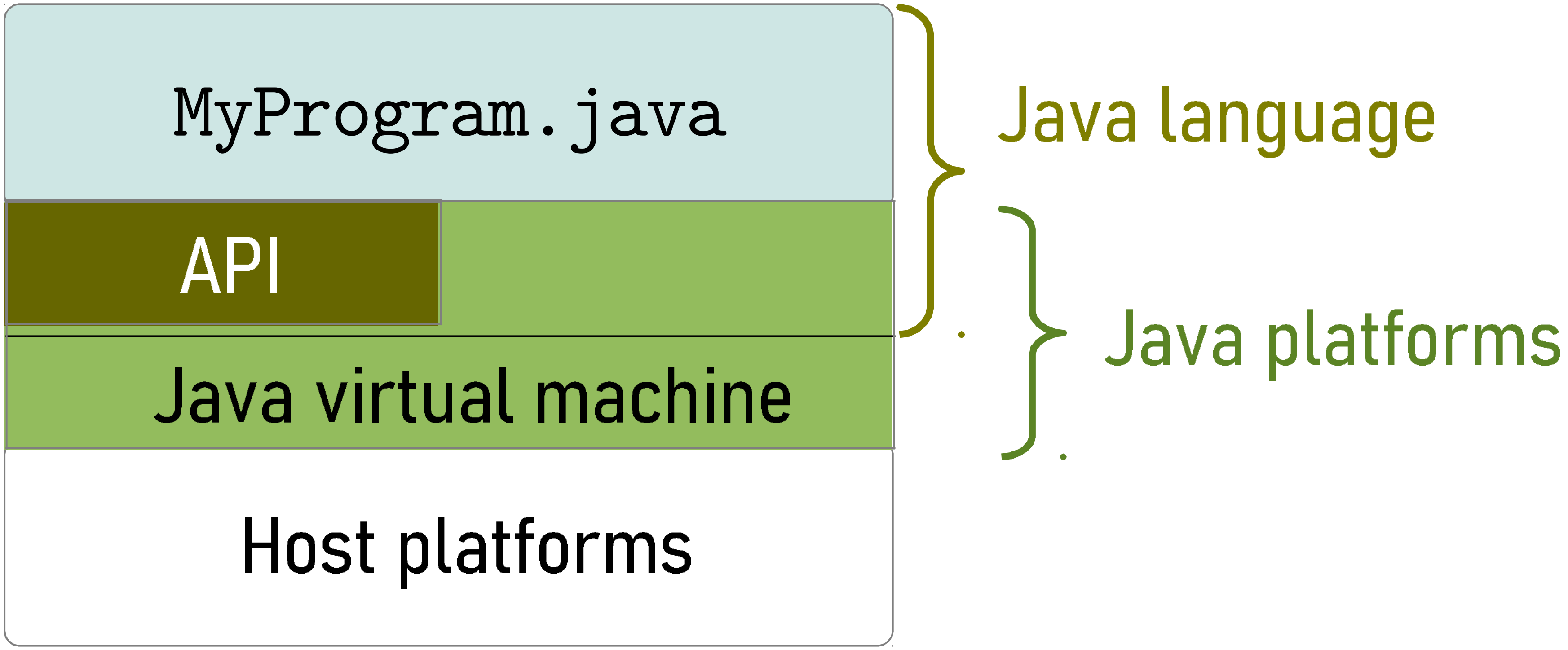
- Basic terminologies
- Java programming language
- Java platform
- Java virtual machine
- New updates on Java 8

Lecture outline

- Basic terminologies
- Java programming language
- Java platform
- Java virtual machine
- New updates on Java 8

1

Basic terminologies



Lecture outline

- Basic terminologies
- Java programming language
- Java platform
- Java virtual machine
- New updates on Java 8

2

Java programming language

- Brief history
- Language features
- The `HelloWorldApp.java` program

Brief history (1)

- **Initial goal:** to build software for networked consumer devices, supporting:
  - Multiple host architectures
  - Secure delivery of software

Brief history (2)

- Similar in syntax to C/C++
  - but omits complex, confusing, unsafe features
- Supported by web browsers via extensions:
  - Java programs are “embedded” in HTML pages
- Design and architecture decisions drew from Eiffel, SmallTalk, Objective C, and Cedar/Mesa

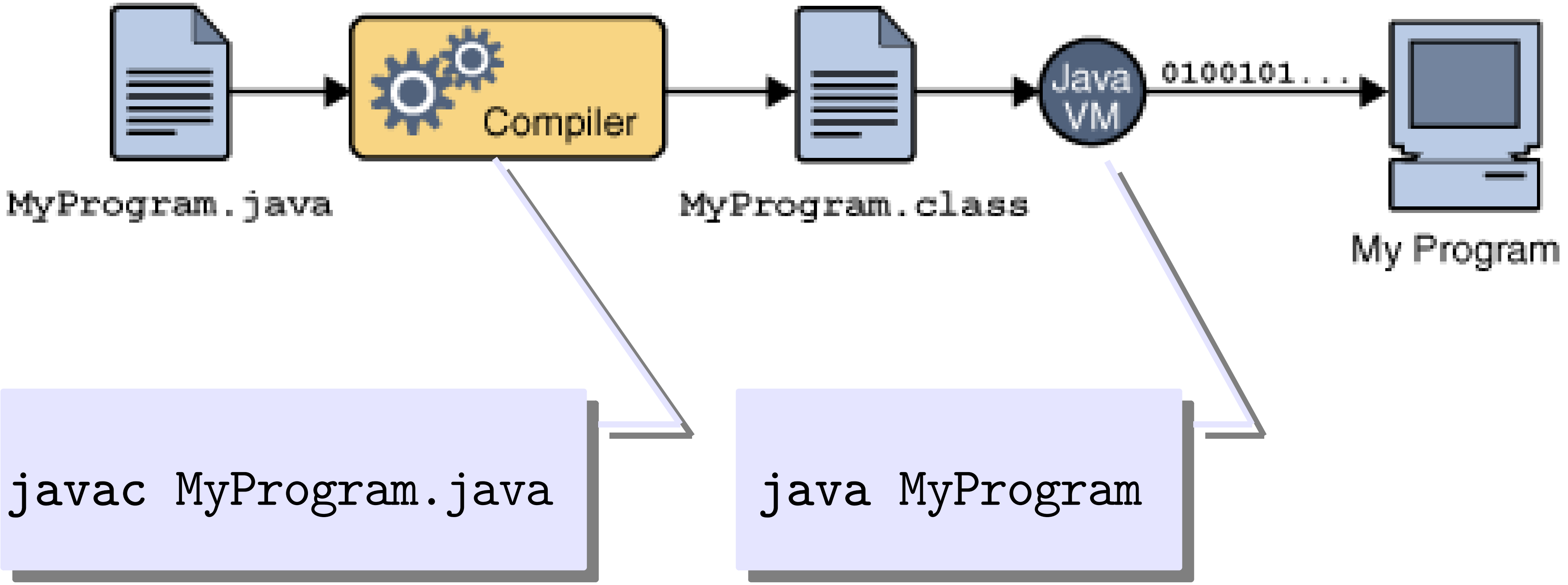
Language features

- Simple, object oriented, familiar
- Robust and secure
- Architecture neutral and portable
- High performance
- Interpreted, threaded, and dynamic

The `HelloWorldApp.java` program

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

Developing a Java program



```
javac MyProgram.java
```

```
java MyProgram
```

An overview of the software development process

Java Application programming interface (API)

- A collection of ready-made components that provide useful capabilities
- Grouped into libraries known as **packages**

Lecture outline

- Basic terminologies
- Java programming language
- Java platform
- Java virtual machine
- New updates on Java 8

Java platform (1)

- A software-based platform in which Java programs run
- Runs on top of other hardware-based platforms
  - e.g. Windows, Linux, etc.

Java platform (2)

- Designed for different classes of host platforms and/or applications
- Examples of host platform classes:
  - Small devices: restricted configuration
  - PCs: standard hardware configuration
  - Servers: high performance configuration
- Examples of application classes:
  - stand alone
  - small scale
  - large scale

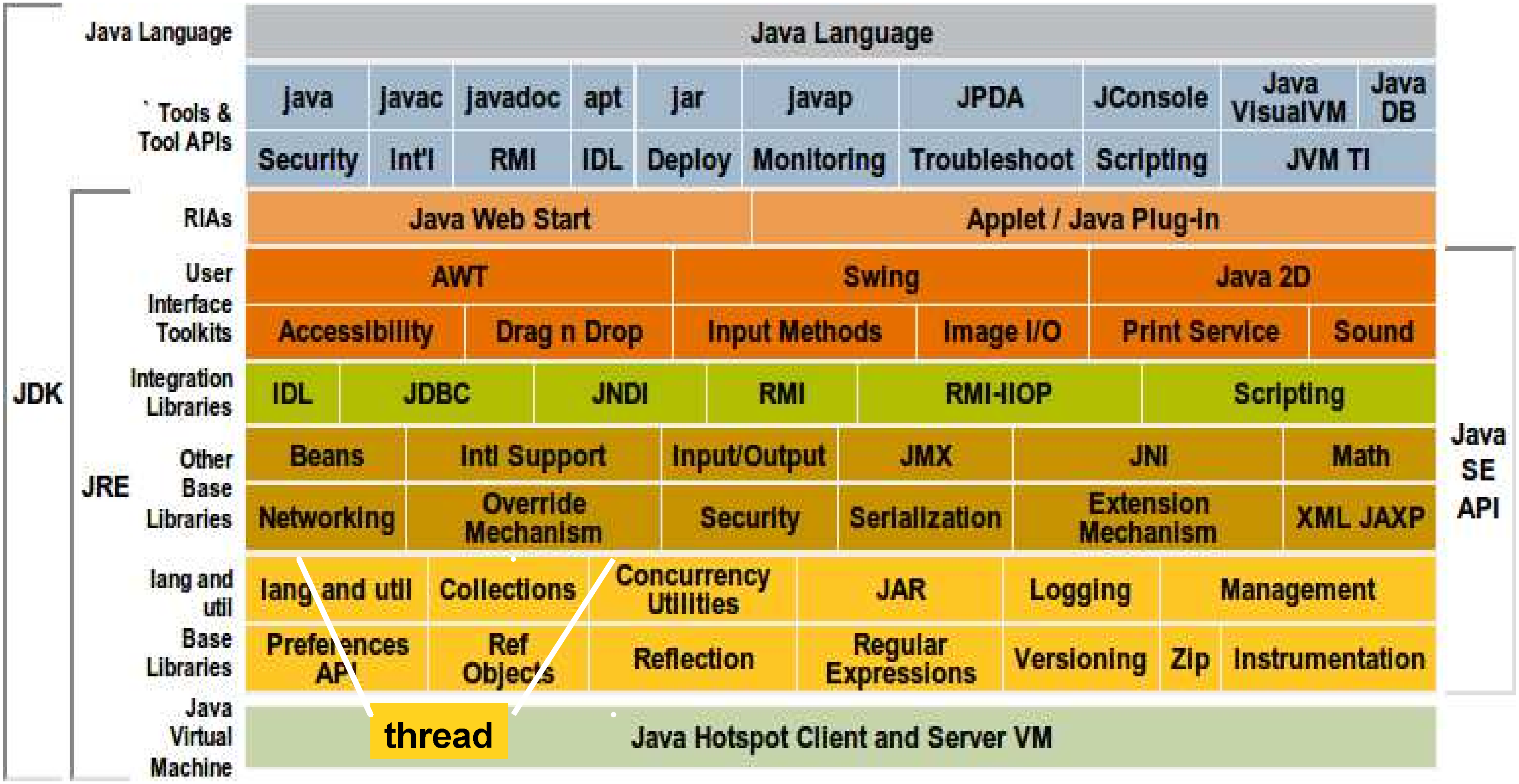
Java platform (3)

- Differ in the JVM implementations and/or API features:
  - Host requirements → Different JVM implementations
  - Application requirements → Different API features
- Three main platforms:
  - **Java SE**
  - Java EE
  - Java ME

Java SE

- Java Standard Edition
- Provides core functionality:
  - basic types and objects
  - programming abstractions for networking, security, database access, GUI development and XML parsing
- Common development tools and deployment technologies

Java SE platform



JDK

JRE

Java SE API

Java Language: Java Language

Tools & Tool APIs: java | javac | javadoc | apt | jar | javap | JPDA | JConsole | Java VisualVM | Java DB

Security | Int'l | RMI | IDL | Deploy | Monitoring | Troubleshoot | Scripting | JVM TI

RIAs: Java Web Start | Applet / Java Plug-in

User Interface Toolkits: AWT | Swing | Java 2D

Accessibility | Drag n Drop | Input Methods | Image I/O | Print Service | Sound

Integration Libraries: IDL | JDBC | JNDI | RMI | RMI-IIOP | Scripting

Other Base Libraries: Beans | Intl Support | Input/Output | JMX | JNI | Math

Networking | Override Mechanism | Security | Serialization | Extension Mechanism | XML JAXP

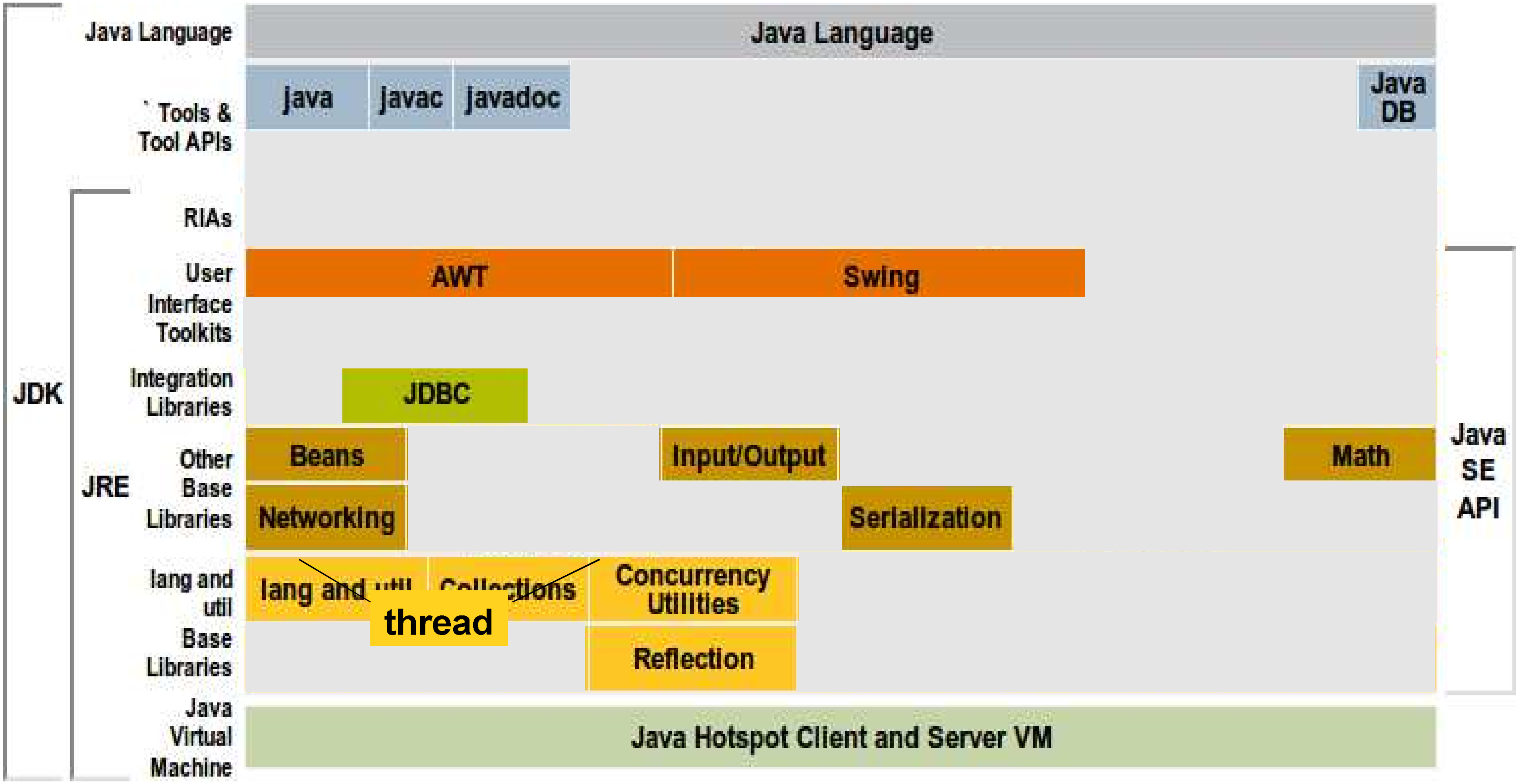
lang and util Base Libraries: lang and util | Collections | Concurrency Utilities | JAR | Logging | Management

Preferences API | Ref Objects | Reflection | Regular Expressions | Versioning | Zip | Instrumentation

Java Virtual Machine: Java Hotspot Client and Server VM

thread

Java SE focus in this module



Java EE

- Java Enterprise Edition
- Built on top of the Java SE platform Designed for:
  - large-scale, multi-tiered, scalable, reliable, and secure network applications
- Provides API and runtime environment

Java ME

- Java Mobile Edition
- Designed for small devices
    - e.g. mobile phones
- Provides API and a small-footprint JVM
- API = subset of Java SE API + libraries for small device applications
- Java ME applications often interact with Java EE platform services

Lecture outline

- Basic terminologies
- Java programming language
- Java platform
- Java virtual machine
- New updates on Java 8

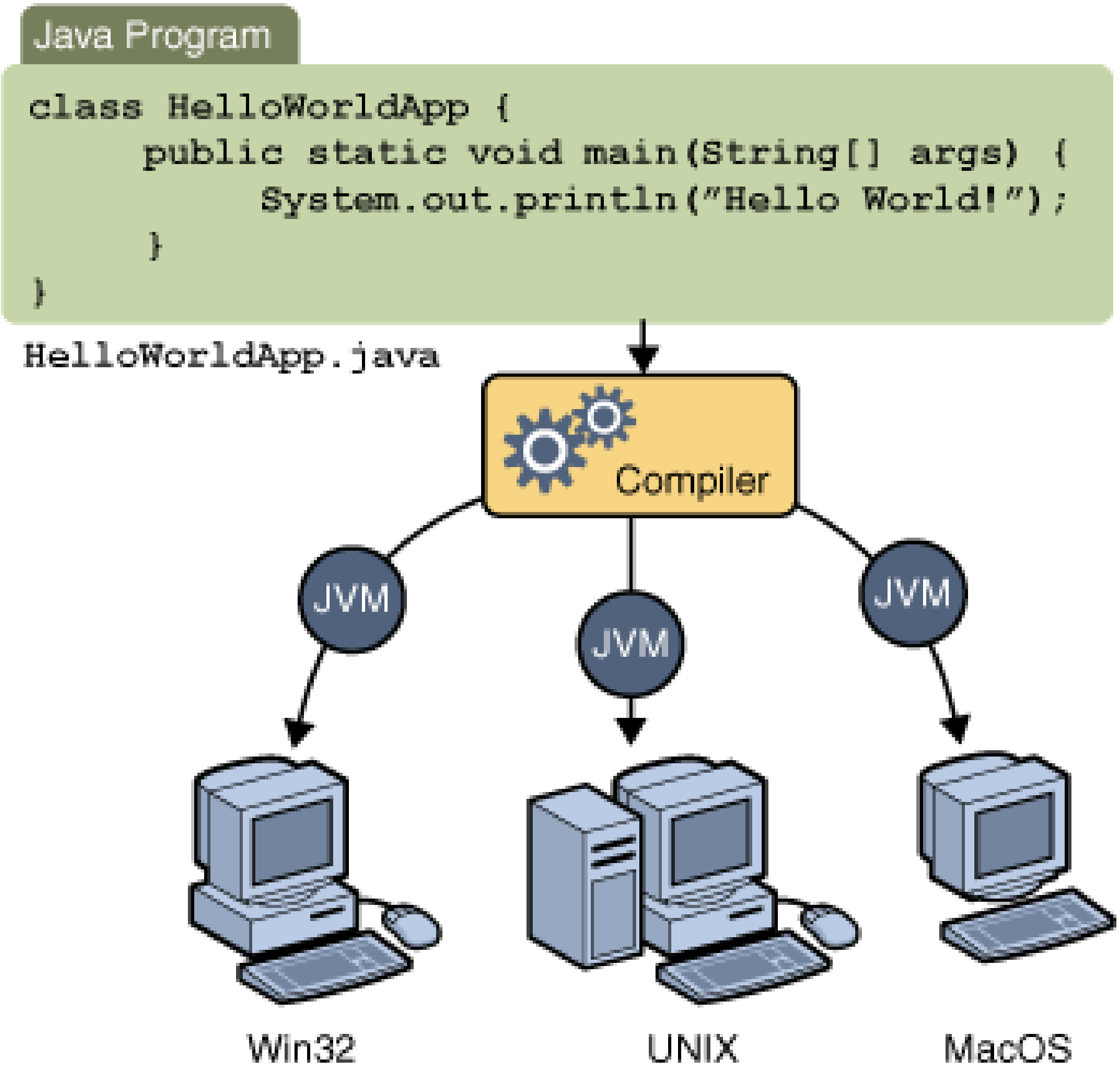
4 Java virtual machine

- Overview and features
- Program execution cycle
- Other selected topics:
  - Stack memory
  - Heap memory

Overview (1)

- The JVM is the base for the Java platform
- Makes Java programs platform-independent:
  - “write once, run anywhere”
- Different versions exist for different hardware-based platforms

Overview (2)



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

JVM Features

- An abstract computing machine with:
  - instruction set
  - memory management
- Emulates the host machines
  - to ensure platform-independent byte codes
- Does not require Java programming language
  - supports any language which can be compiled into the **class file format** (Java bytecode)

Program execution cycle

- Virtual machine start up
- Loading
- Linking
- Initialisation
- Creation of new class instances
- Finalization of class instances
- Unloading
- Virtual machine exit

Virtual machine start up

- The method `main` is invoked with argument `String[]`:
  - Header: `public static void main (String[])`
  - argument is a nullable `String` array
- Invocation is typically from the command line:

```
java HelloWorldApp say hello world!
```

HelloWorldApp with arguments

```java
public class HelloWorldApp {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```

Loading

- Class `HelloWorldApp` is loaded by `ClassLoader`
- The loaded class is an object of class `Class`
  - cached for subsequent use
- Loading may fail due to:
  - incorrect class file format
  - incorrect version of the class file format
  - not found

Linking

- Combines the loaded class into the runtime state of the JVM
- Three steps:
  - **Verification:** check the class structure
  - **Preparation:** creating and initializing class fields to default values
  - **Resolution:** resolve references to other classes

Initialisation

- Execute initialisers:
  - class (static) initialisers
  - initialisers for static fields
- Also causes any super class(es) to be loaded, linked and initialised:
  - if not already

Creation of new class instances

- Objects are created if required
- Object creation involves:
  - allocating enough memory space for all variables (declared in the class and super class)
  - initialising the variables executing a constructor method

HelloWorldApp with objects

```java
public class HelloWorldApp2 {
    public static void main(String[] args) {
        String msg = "Hello world!";
        // or String msg = new String("Hello world!");
        System.out.println(msg);
    }
}
```

Finalization of class instances

- This mechanism is deprecated as of JDK 9
- Remove objects that are no longer in use
- Java garbage collector automatically remove these objects
- Finalizers are used to prepare objects for removals
- Overrides `Object.finalize`

Finalization example

```java
public class FinalizeExample {
    public static void main(String[] args) {
        FinalizeExample obj = new FinalizeExample();
        System.out.println(obj.hashCode());
        obj = null;
        System.gc(); // calling garbage collector
        System.out.println("end of garbage collection");
    }

    @Override
    protected void finalize() {
        System.out.println("finalize method called");
    }
}
```

Unloading

- Unload unused classes to reduce memory use
- A class is unloaded when its associated `ClassLoader` is removed
- System classes may never be unloaded

Virtual machine exit

- When one of two things happens:
  - all non-daemon processes (threads) finish execution
  - `System.exit()` or `RunTime.exit()` is invoked

Stacks

- Each program thread has a stack to:
  - hold local variables and partial results
  - used in method invocation and return
- Stack-overflow error is thrown if a stack runs out of memory
- Stack size may be changed via JVM options

Specifying thread stack size

- To specify a 1M stack size from the command line:

```
java -Xss1M HelloWorldApp
```

Heap

- A run-time memory shared among all JVM threads:
  - created on JVM start up
- Used for storing objects
- Heap space is reclaimed by a garbage collector
- Out-of-memory error is thrown if heap runs out of space
- Heap size may be changed via JVM options

Heap

- To specify initial and max heap sizes from command line:

```
java -Xms256M -Xmx256M HelloWorldApp
```

-Xms: the initial size

-Xmx: the maximum size

Lecture outline

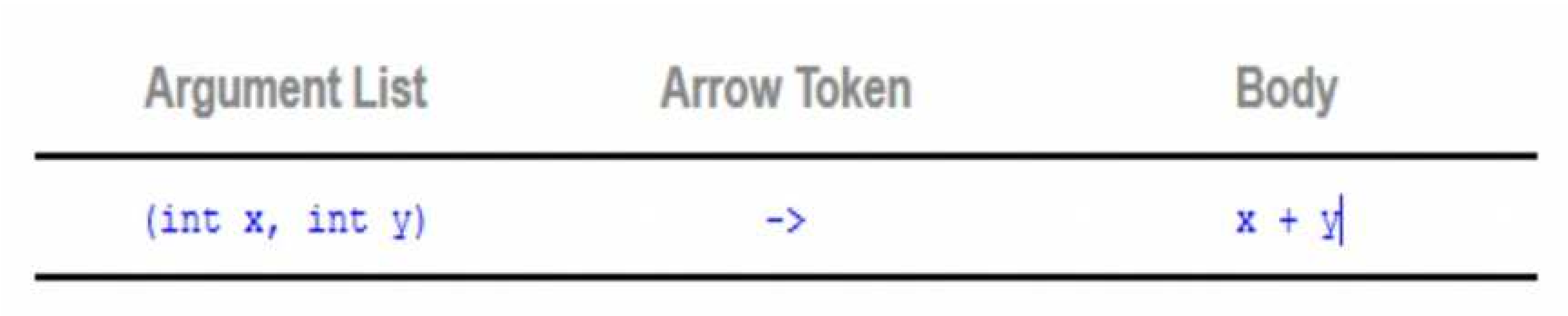
- Basic terminologies
- Java programming language
- Java platform
- Java virtual machine
- New updates on Java 8

5 Updates on Java 8

- Lambda Expression Date and Time API
- Nashorn JavaScript Engine
- And some other “headache” things

Lambda Expression

- Provide a clear and concise way to represent one method interface using an expression
- Syntax:



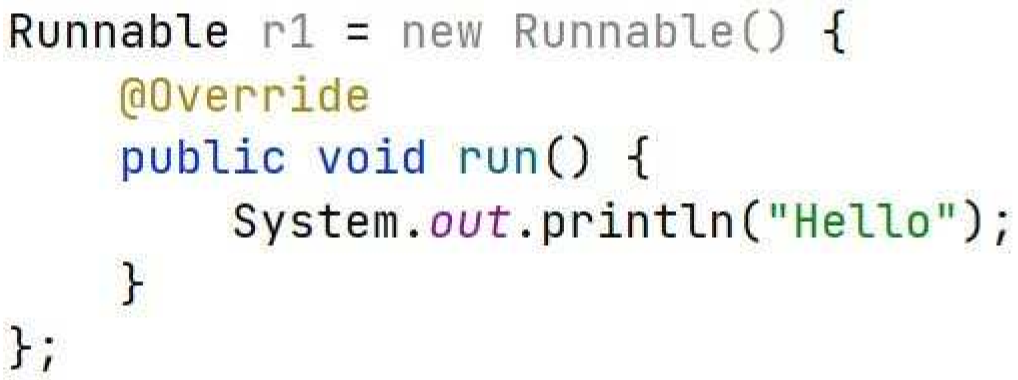
Argument List	Arrow Token	Body
`(int x, int y)`	`->`	`x + y`

Lambda Expression

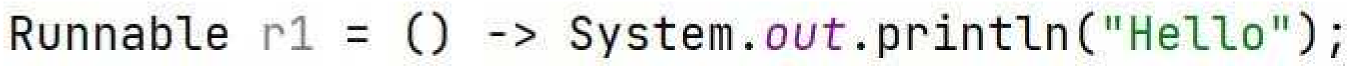
- Samples:
  - (int x, int y) ->x+y : takes two integer arguments, named x and y, and uses the expression form to return x+y
  - () ->42 : takes no arguments and uses the expression form to return an integer 42
  - (String s) -> {System.out.println(s)}: takes a string and uses the block form to print the string to the console, and returns nothing

Lambda Expression

- **Runnable** using Lamda:



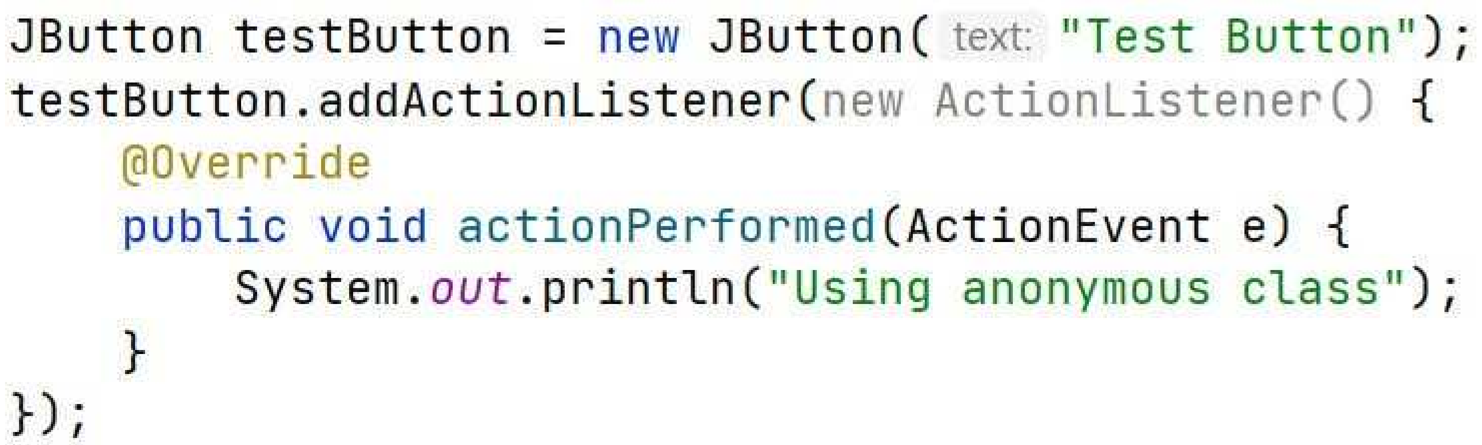
```
Runnable r1 = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello");
    }
};
```



```
Runnable r1 = () -> System.out.println("Hello");
```

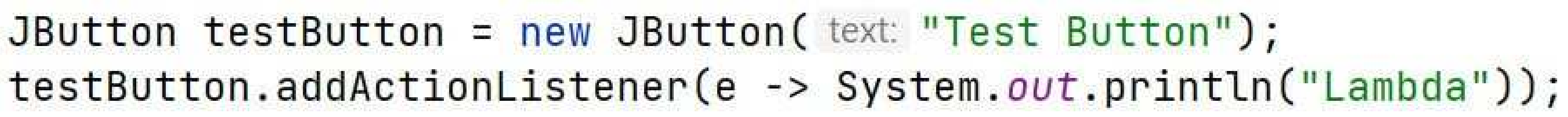
Lambda Expression

- Listener Lambda



```java
JButton testButton = new JButton( text: "Test Button");
testButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Using anonymous class");
    }
});
```

⇩



```java
JButton testButton = new JButton( text: "Test Button");
testButton.addActionListener(e -> System.out.println("Lambda"));
```

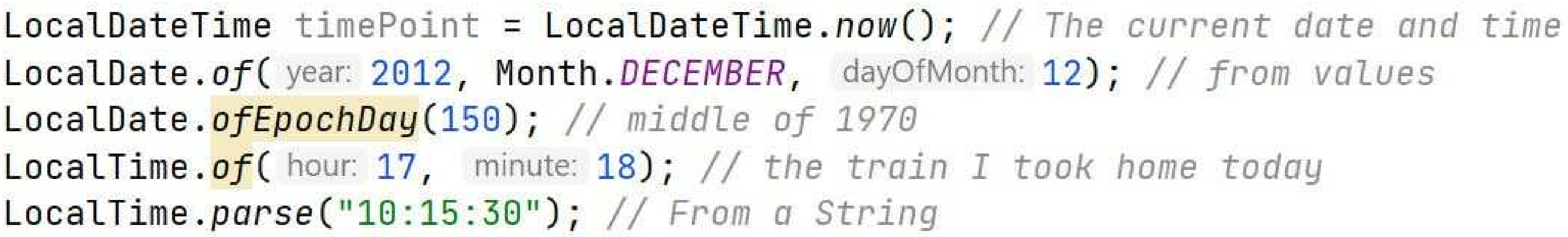
Date and time API

- Why we need new date & time library?
  - Inadequate support for the date and time use cases of ordinary developers
  - Some date and time classes also exhibit quite poor API design (e.g: years in java.util.Date start at 1900, months start at 1, and days start at 0—not very intuitive)
  - Existing classes (such as java.util.Date and SimpleDateFormatter) aren’t thread-safe, leading to potential concurrency issues for users

Date and time API

- The new API is driven by three core ideas:
  - Immutable-value classes
  - Domain-driven design
  - Separation of chronologies
- Changes in details:

http://www.oracle.com/technetwork/articles/java/jf14-date-time-2125367.html



```
LocalDateTime timePoint = LocalDateTime.now(); // The current date and time
LocalDate.of( year: 2012, Month.DECEMBER,  dayOfMonth: 12); // from values
LocalDate.ofEpochDay(150); // middle of 1970
LocalTime.of( hour: 17,  minute: 18); // the train I took home today
LocalTime.parse("10:15:30"); // From a String
```

Nashorn JavaScript Engine

- New JavaScript engine developed in the Java programming language by Oracle
- Goal: To implement a lightweight high-performance JavaScript runtime in Java with a native JVM
- Embed JavaScript in a Java application and also invoke Java methods and classes from the JavaScript code

Nashorn JavaScript Engine

- By using Nashorn the developer can perform the magic of:
  - Running JavaScript as native Desktop code
  - Using JavaScript for shell scripting
  - Call Java classes and methods from JavaScript code

Summary

- Java technology includes Java language, platform, virtual machine and API
- Java language is object oriented, robust, and architecture neutral
- Different types of Java platforms designed for different classes of hosts and applications
    - Java SE, EE, ME
- Java virtual machine is a software abstraction of the host, making Java programs platform independent
    - Programs are executed in a cycle

Summary

- Some new updates on Java 8:
  - Lamda expression
  - Date and time API
  - Nashorn JavaScript Engine