Java - Lesson 1

Introduction

Kirill Volkov @ MERA
github.com/vulko/JavaBasicsLectures

Java Environment

- Sun Microsystems released Java 1.0 in 1995
- WORA (write once, run anywhere) principle
- Performance is slower compared with C/C++
- Automatic memory management (garbage collection), Cpp-like syntax
- Object oriented, class based, concurrent

```java
class HelloWorldApp {

    public static void main(String[] args) {

        System.out.println("Hello World!"); // Prints the string to the console.

    }

}
```

Java Environment

IDE: Eclipse, NetBeans, IntelliJ Idea

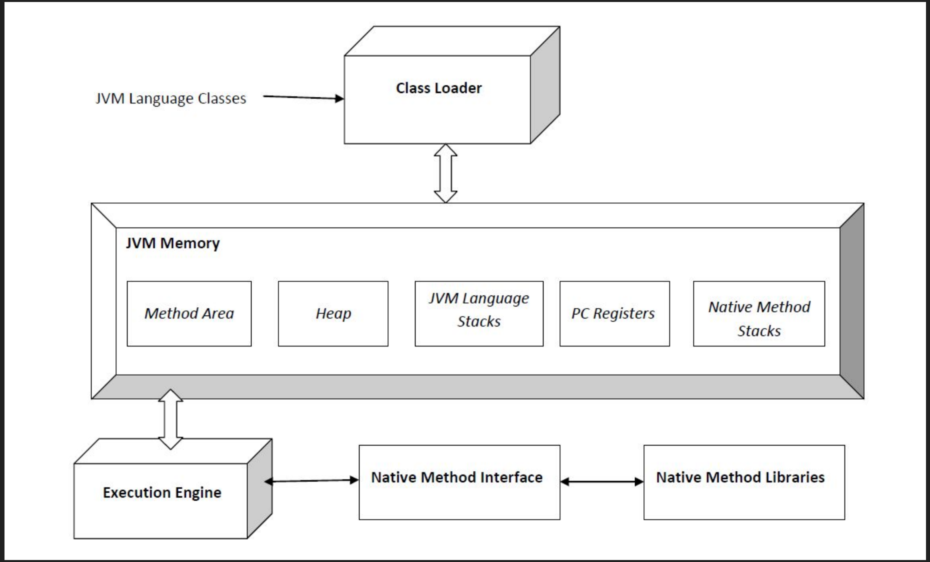
JDK (Java Development Kit):

java, javac, javadoc, jar, debugger, libs

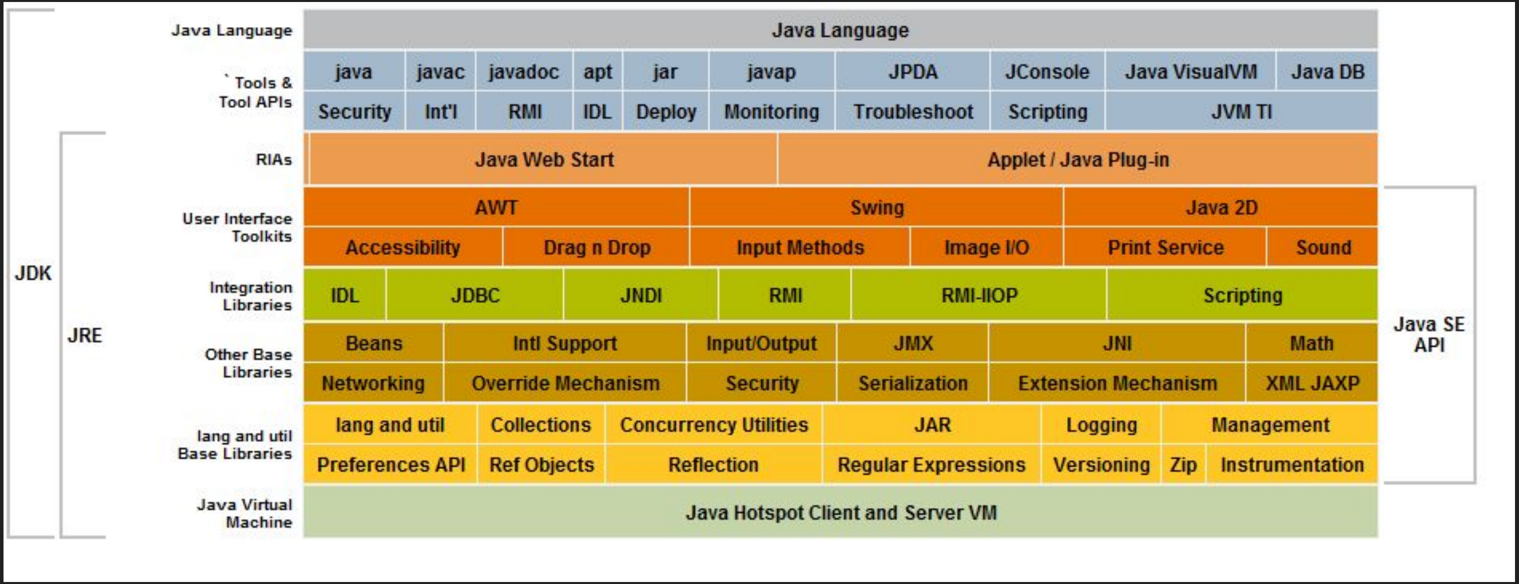
JVM runs Java Byte Code

*.java -> *.class

Jar packages



Java SE Platform



Garbage Collection

- Works as a part of Virtual Machine all the time
- Manages heap
- Tracks objects lifetime, manages objects as groups
- Marks objects that should be removed, removes objects that are not used
- System might be notified to cleanup memory, no way to remove objects manually to free up space
- JVM has several algorithms of garbage collection

Java data types

boolean	true OR false	false
byte	1-byte, signed (-128..127)	0
short	2-byte, signed	0
int	4-byte, signed	0
long	8-byte, signed	0L
float	4-byte, floating point	0.0f
double	8-byte, floating point	0.0d
char	2-byte Unicode character	'\u0000'
String	4-byte	null
Object	4-byte	null

Package

- Package is a collection of classes
- Default package
- Naming convention ParentPackage.ChildPackage
- Strict source code structure on filesystem

```
package JustAnotherPackage;

public class JustAnotherClass {/* code */}
```

```
package testPackage;

import JustAnotherPackage;

public class TestClass { JustAnotherClass object = new JustAnotherClass(); }
```

Class

- Class is a basic instance of OOP principle
- Class is an abstraction
- Each *.java source file represents a class
- Object is an instance of a class, created with a **new** keyword, that triggers a class object constructor.

```
public class TestClass {

    public void doSomething() {

        Object object = new Object();

    }

}
```

Class constructors, methods and field

- Default constructor without parameters
- Constructor with parameters
- Class-method
- Class-field

```
public class TestClass {

    modifiers TYPE name;



    modifiers TestClass() {}

    modifiers TestClass(int param1, boolean param2) {}

    modifiers returnType methodName() {}

}
```

Modifiers

Visibility modifiers (Encapsulation):

- default (not specified)
- private
- protected
- public

Other modifiers:

- final
- static
- abstract

Static-modifier

Static fields:

- Same value for each instance of a class
- Exist without creating an instance
- Can be accessed without creating an instance
- JVM creates when class is accessed for the first time

```
public class TestClass {

    public static int SOME_INT = 10;

}
```

```
System.out.println(“Static int value = ” +TestClass.SOME_INT);
```

Static-modifier

Static methods:

- Can trigger only static methods
- Can access only static variables
- Can’t use ***this*** and ***super***

```
public class TestClass {

    private static int SOME_INT = 10;

    public static int getStaticInt() { return SOME_INT; }

}
```

```
System.out.println(“Static int value = ” +TestClass.getStaticInt());
```

Static-modifier

Static blocks:

- Triggered by class-loader only once
- Mostly used to initialize static collections

```
public class TestClass {

    private static int SOME_INT = 10;

    static {

        SOME_INT = 20;

    }

}
```

Final-modifier

- Final variable can’t be modified
- Final method can’t be overridden
- Final class can’t get nested

```java
public final class TestClass {          // can’t be a parent to any other class

    private final int SOME_CONST = 10; // constant value

    private final int getSameName() {} // can’t override

}
```

Abstract-modifier

- Implements polymorphism OOP principle
- Abstract method is a method without implementation
- If a class contains at least one method, has to be marked as abstract
- Abstract class can’t be instantiated
- Abstract methods have to be implemented in nested classes

```
public abstract class TestClass {          // can’t be instantiated

    public abstract int getValue();        // has to be implemented in nested classes

}
```

Object

- Any non-primitive type in Java is implicitly nested from Object

protected Object	**clone**()	void	**notify**()
boolean	**equals**(Object obj)	void	**notifyAll**()
protected void	**finalize**()		
Class<?>	**getClass**()	void	**wait**()
int	**hashCode**()	void	**wait**(long timeout)
String	**toString**()	void	**wait**(long timeout, int nanos)

this and super references

- Each object contains a link to self and super class

```
public class Parent {

    public int getValue() { return 5; }

}
```

```
public class Child extends Parent {

    public int getValue() { return 10; }

    public int getParentValue() { returnsuper.getValue(); } // returns 5

    public int getChildValue() { returnthis.getValue(); }    // returns 10

}
```

this and super references

- Can be used in constructor

```
public class Parent {

    private String mName;

    public Parent(String name) { mName = name; }

}
```

```
public class Child extends Parent {

    public Child() { this(“empty name”, 12345); }

    public Child(String name) {super(“child”); }

    public Child(String name, int someValue) {super(name); }

}
```

Inner (local) class

Inner class is a class defined inside of another class

Good for:

- Due to convenience reasons (access of wrapper/inner class methods and variables, declaring instance that can’t be used without wrapper instance)
- To avoid class being visible from within the package
- To implement some functionalities used within wrapping class

```
public class SomeParent {

    public static enum Type { TYPE_1, TYPE_2, TYPE_3 }

    public class SomeInnerClass {}

}
```

Inner class types

- Inner non-static class is called <u>inner class</u>
  - Member inner class
  - Local class
  - Anonymous class

Can access all variables and methods of an outer class

Can reference outer class instances

- Inner static class is called <u>static nested class</u>

Interface

Interface is similar to a pure abstract class.

A class can extend only 1 parent, but implement multiple interfaces.

Class implementing an interface must implement all of it’s defined methods.

```
public interface SomeInterace {

    public static int ID = 1;

    void doSomething();

    int getSum(int a, int b);

}
```