
Shaping
ideas
into great
Software
Products
BELATRIX

Fundamentals of Java Programming

Belatrix SF

Goals

- Give a general view of Java as one of the most popular programming languages and software developing platform.
- To know in detail, what are and how to use the most basic elements of Java like data types, operators, methods, classes, control structures and so on.
- To know what is the Java Virtual Machine (JVM), and how it works.

Syllabus

- What is Java?
  - Definition
  - Evolution
  - Usage
- Setup / Requisites
  - JDK / JRE
  - IDE (Eclipse and Intellij IDEA)
  - Workspace / Project

Syllabus

- Starting with Java
  - Primitives
  - Classes
  - Attributes and methods
  - Packages
  - Enums
- Object-Oriented Programming
  - Objects
  - Inheritance / Polymorphism
  - Access modifiers
  - Abstract classes
  - Interfaces

Syllabus

- Control structures
  - Conditionals
  - Loops
  - Collections
  - Exceptions
- Unit testing
- Java Virtual Machine (JVM)
  - Definition
  - How it works
  - Compilation / Execution
- Singleton Pattern

Agenda

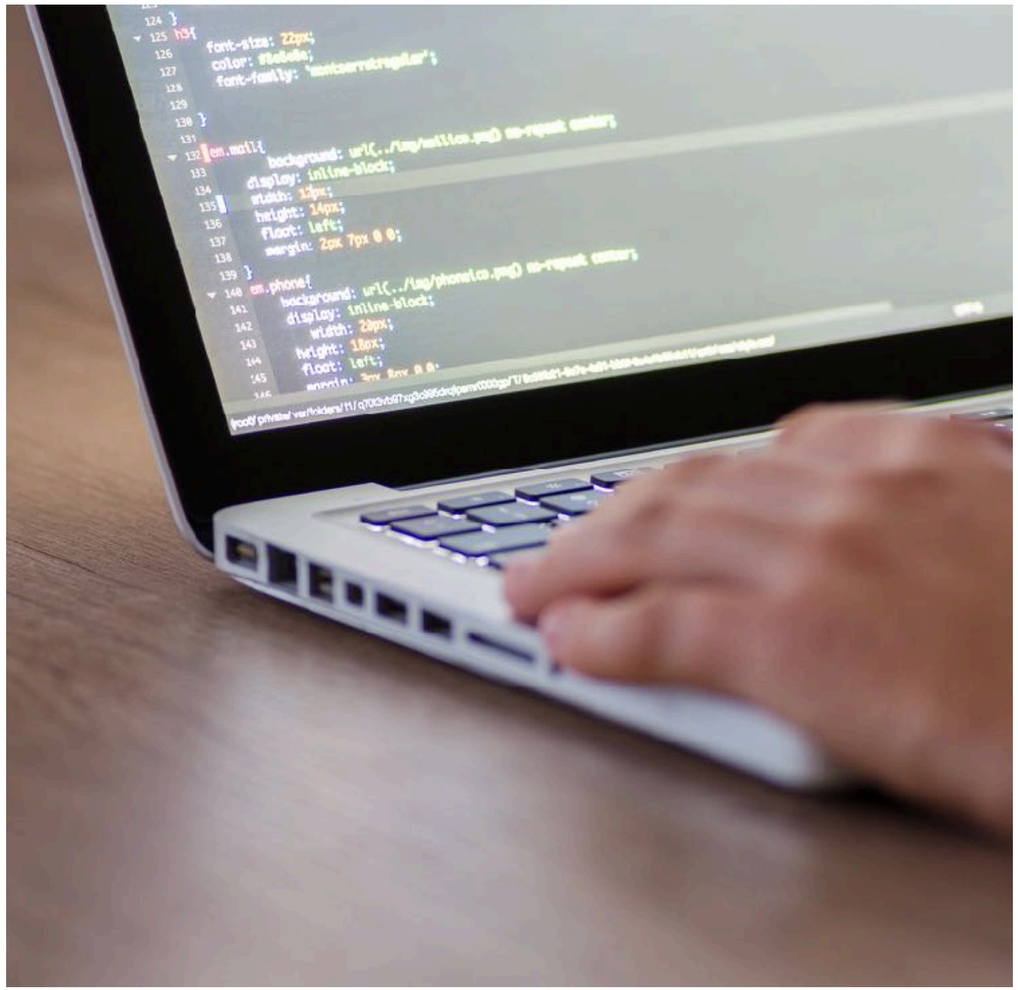
- Session #1
  - General concepts
  - What is Java? Why to use it?
  - Java Programming 101 (Basics): Datatypes, Classes, Methods.
  - Provisioning for Java Development: IDE, Workspace, JDK Setup
- Session #2
  - Object-Oriented Programming
  - Java Programming: Control Structures I (Conditional, Loops, Collections)
- Session #3
  - Java Virtual Machine (JVM). JRE/JDK.
  - Java Programming: Exceptions
- Session #4
  - Singleton
  - Unit testing
  - Evaluation

Getting into context …

Fundamentals of Java Programming

What is Programming?

- “Programming is the art of telling another human what one wants the computer to do” - Donald Knuth
- “Computer programming is a way of instructing electronic machines to perform tasks, solve problems and provide human interactivity” - @YearOfCodes
- To tell instructions to devices so they perform tasks.

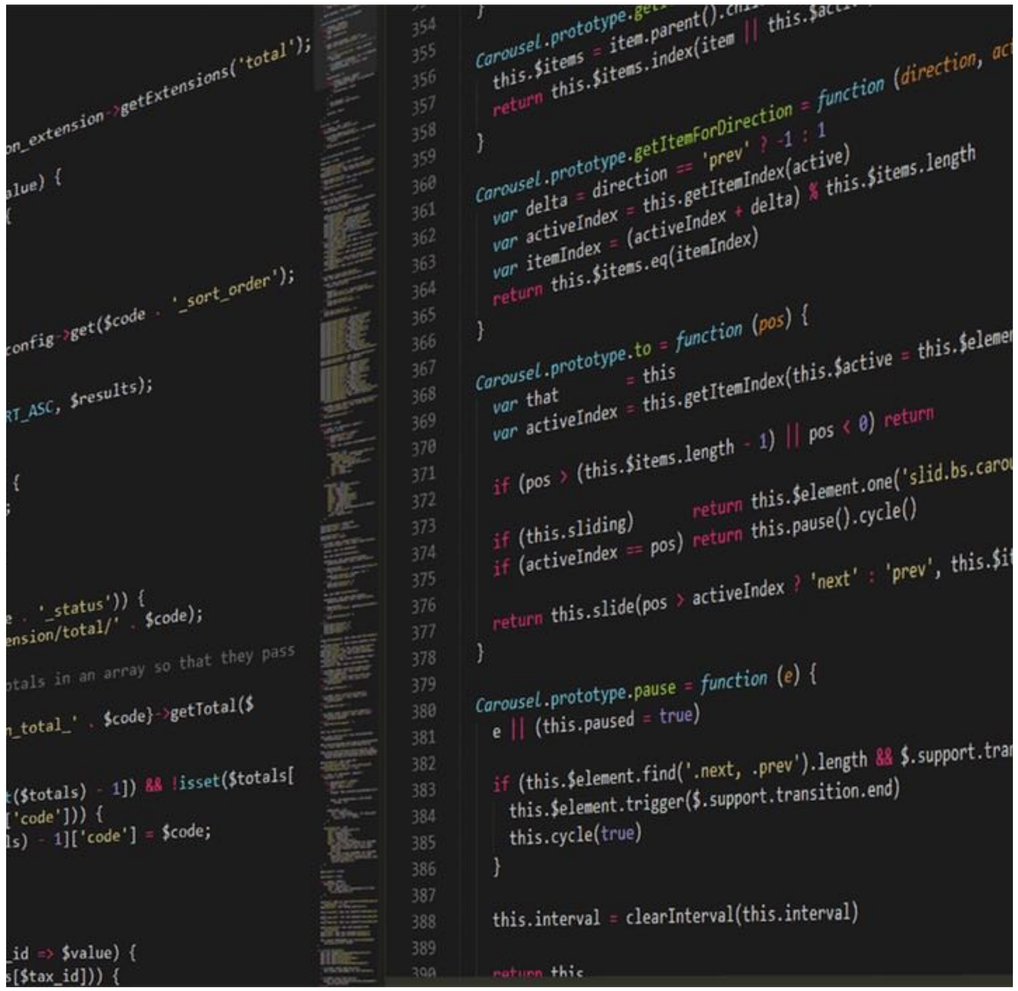


Programming Language

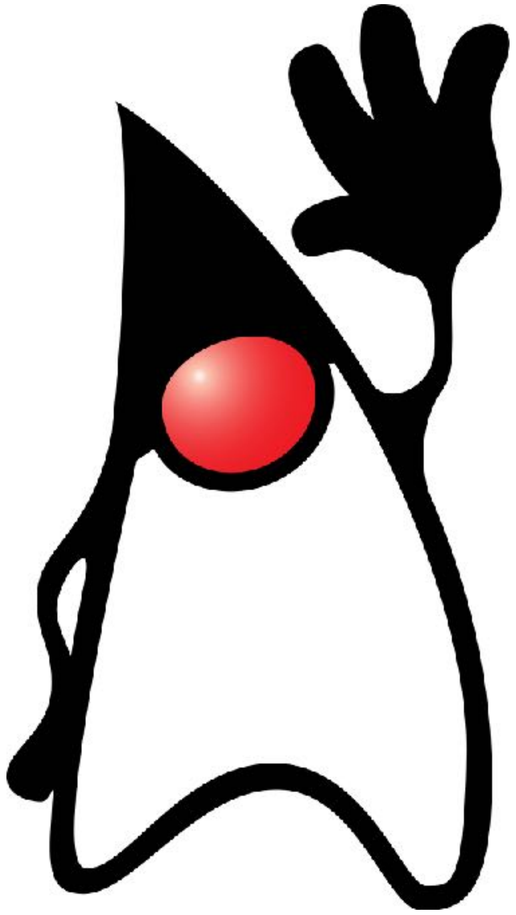
- “Programming language is very specific to instructing a computer to do a particular structure of a sequence. It's the very way you tell the machine what you want it to do.” - Brian Kernighan
- Set of operator and instructions, with rules or syntax on how they should be placed, that allow to make computers to complete the tasks that are needed.
- To express programs
- Machine code: fast, efficient, specific CPU tasks, binary, depends on the device architecture, complicated.

Programming Languages: Types

- Compiled vs Interpreted
- Compiled languages are usually faster at execution time than Interpreted ones.
- Syntax is simpler for interpreted languages, faster and easier to develop.
- Low level: Closer to machine code.
- High level: More natural



What is Java?

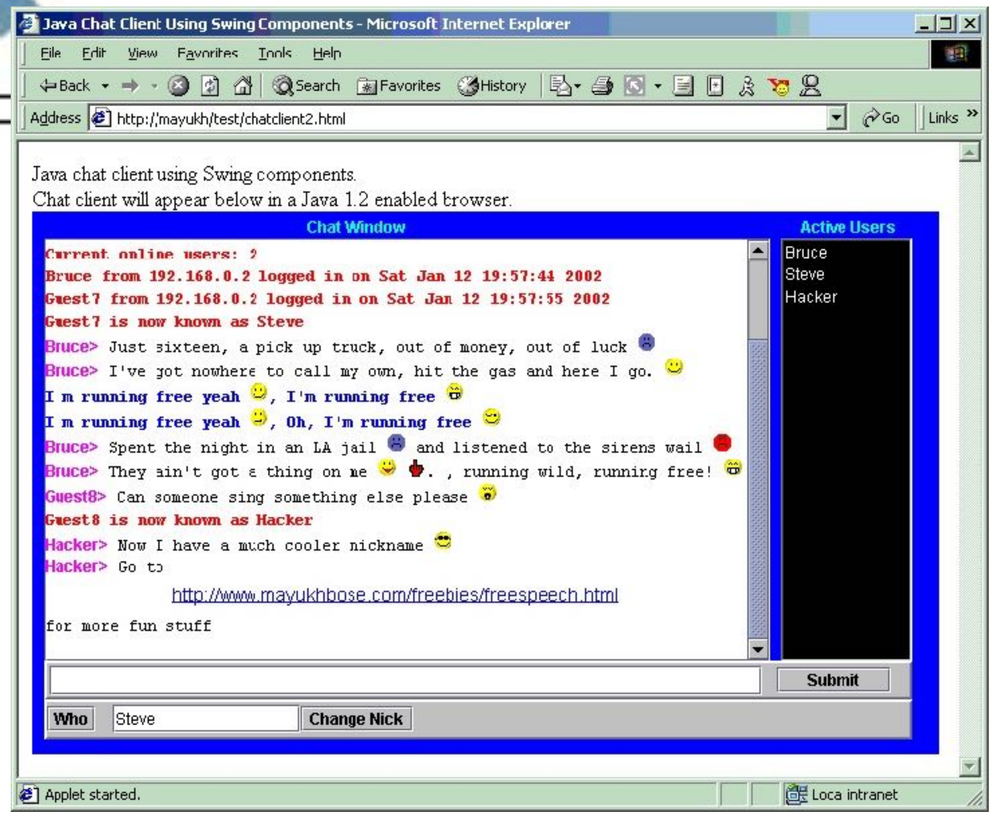


What is Java?

- High-level, compiled programming language.
- Conceived by James Gosling (Lead Designer) in Sun Microsystems on 1991.
- Though to be a platform-independent (“architecture-neutral”) language.
- Popularized by WWW late 90’s growth.

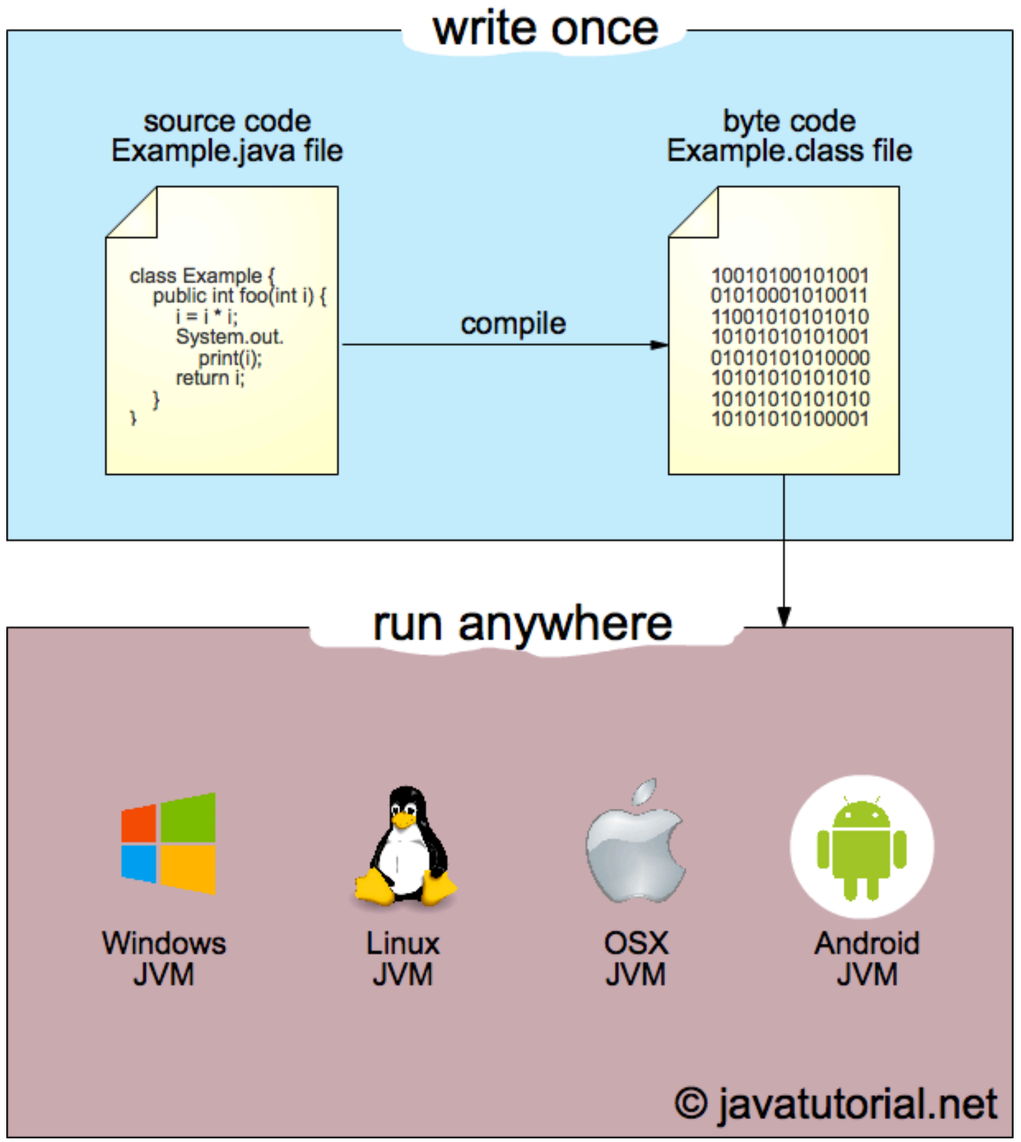
Java Applets

- Designed to be distributed through Internet
- Java Swing.
- Small programs similar to desktop apps.
- Dynamic and self-executing on Web browser.
- Security
- Portability

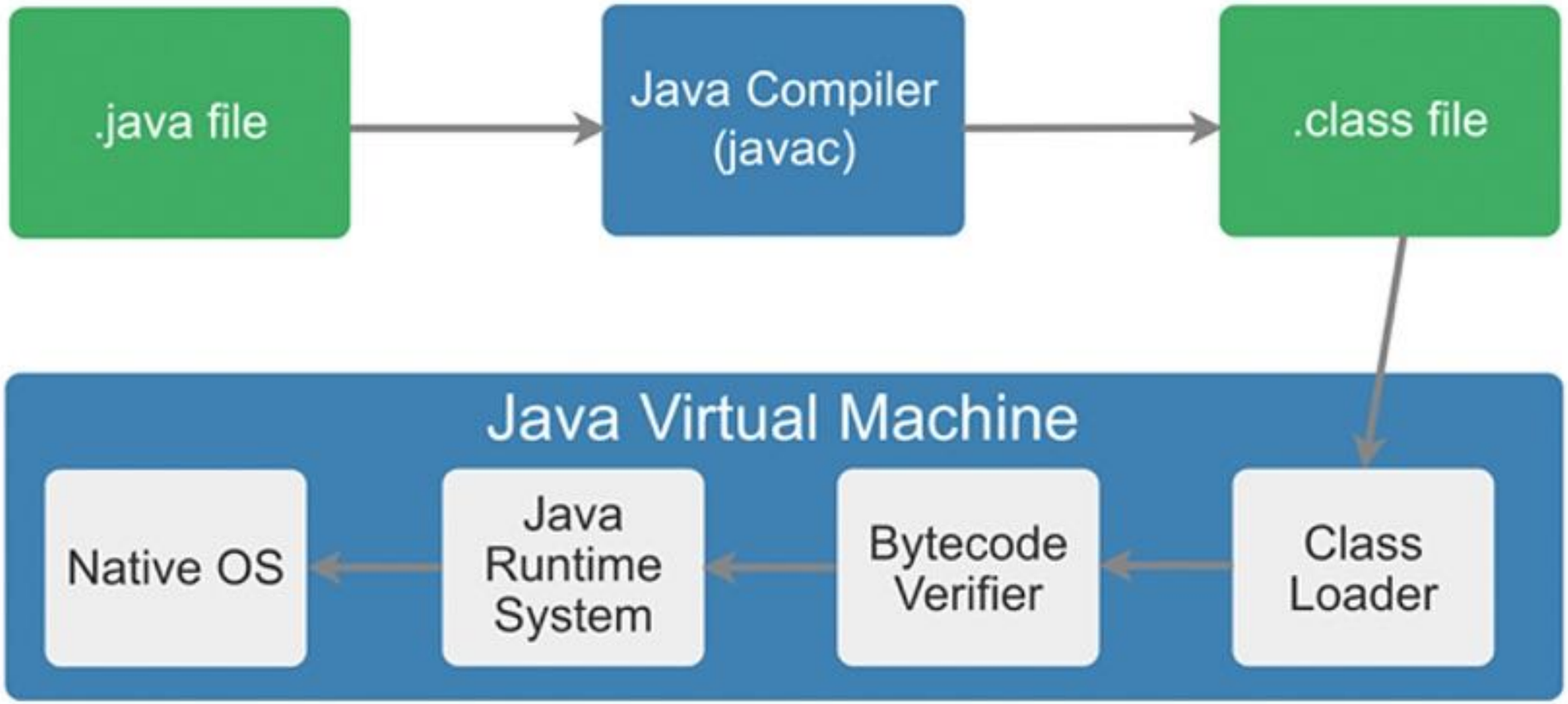


Bytecode

- Magic behind Java portability
- Compiler output
- Not executable code by itself
- Run by Java Virtual Machine (JVM)

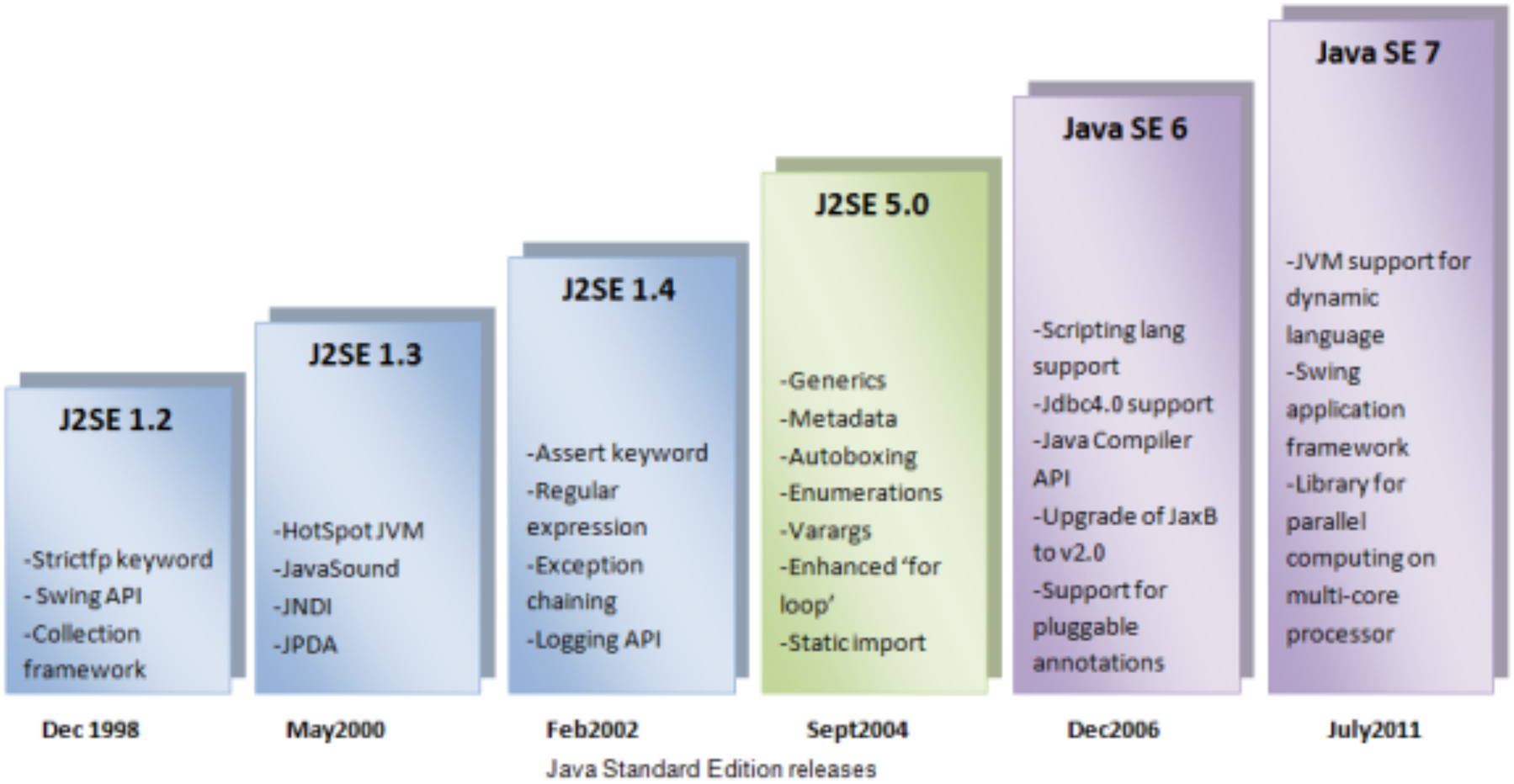


What is Java Bytecode?



Source: http://www.techlila.com/write-programs-linux/

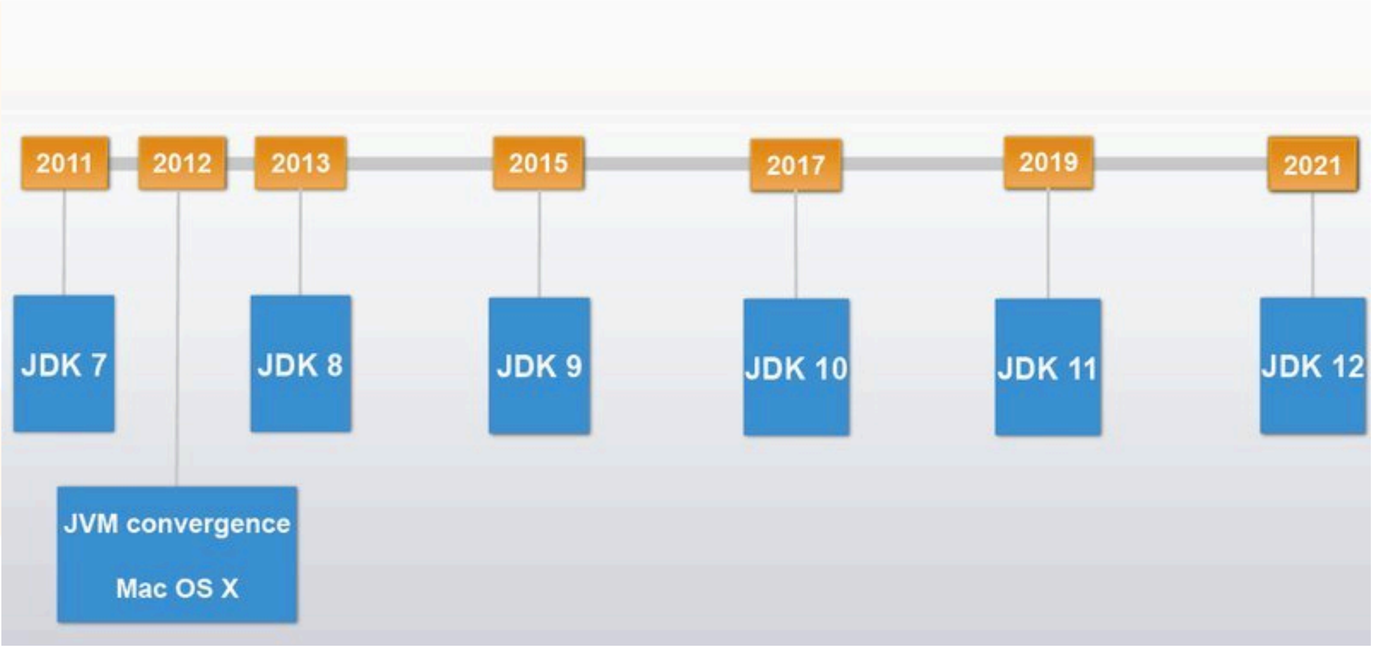
Evolution



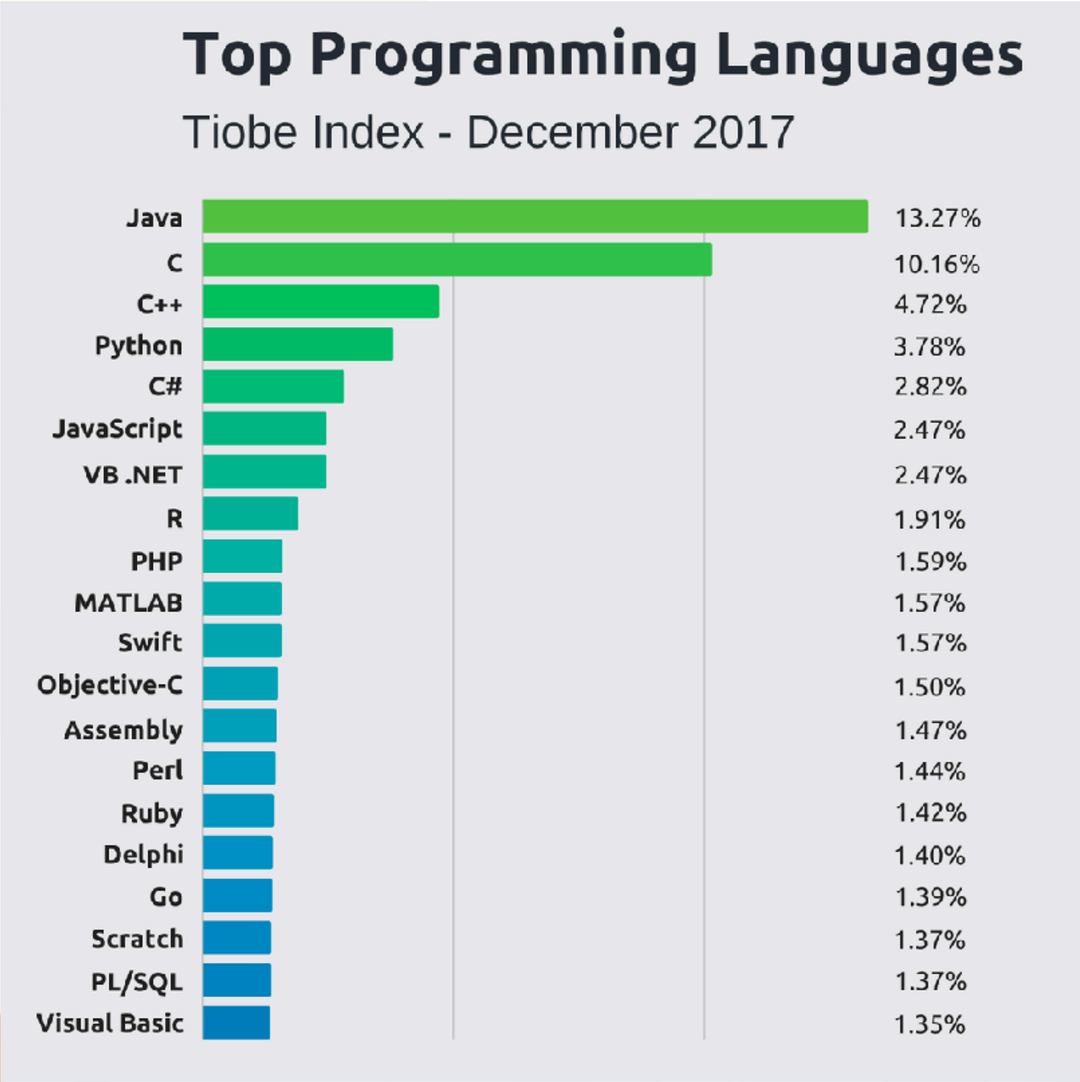
Java Standard Edition releases

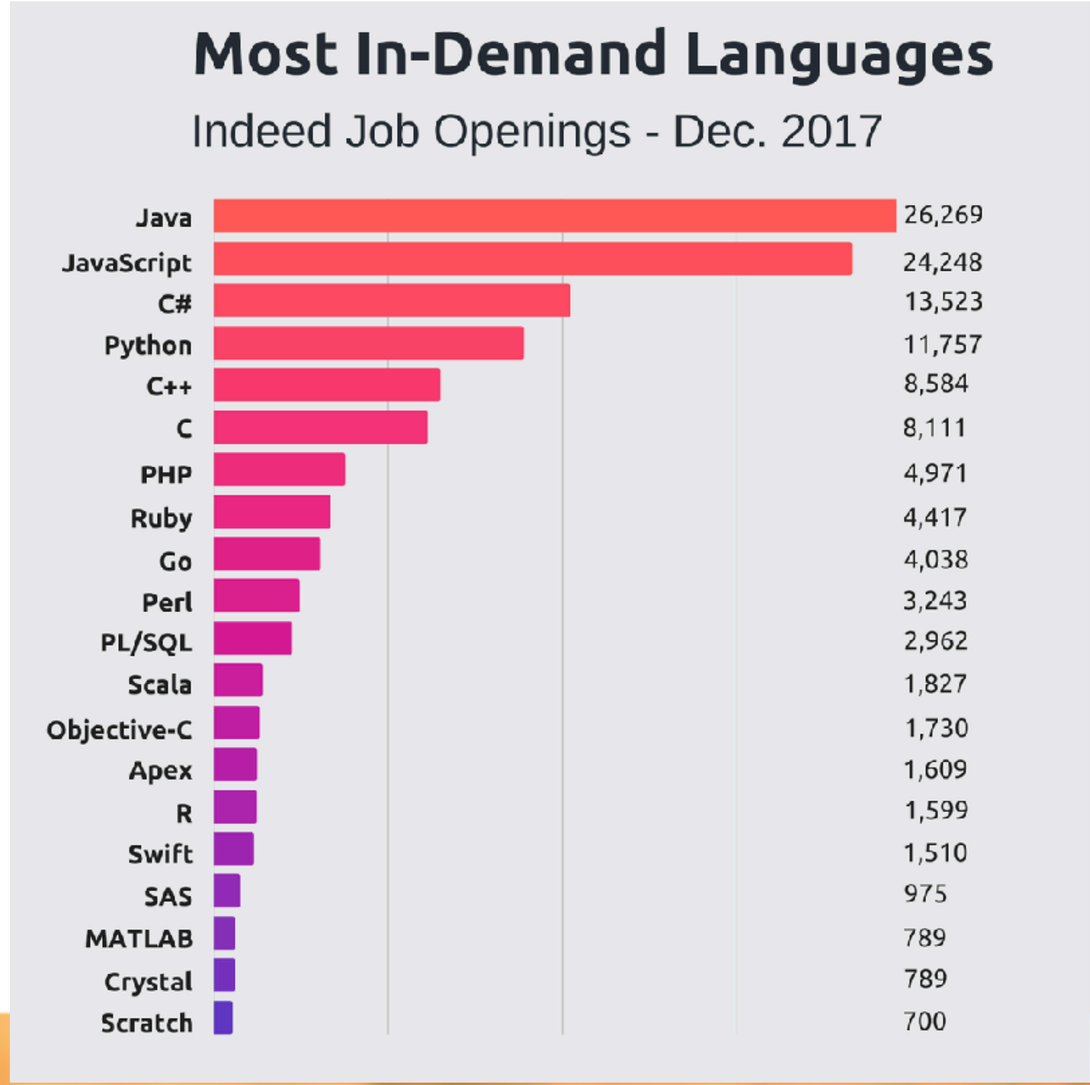
Evolution

Java SE 2012 to Java 12

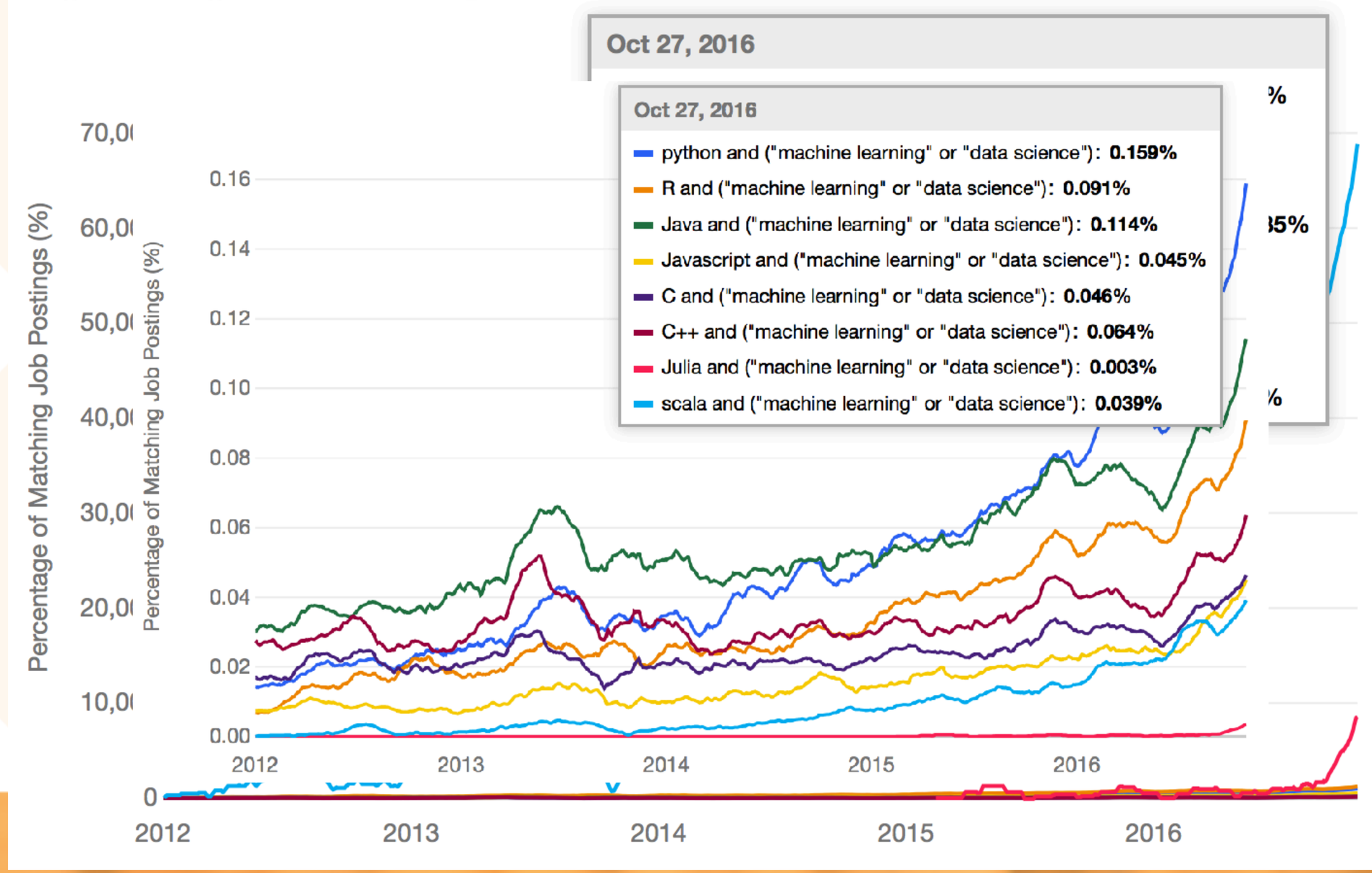


Why to learn Java?





Languages requested for ML and DS



How to start with Java?

Provisioning for Java development

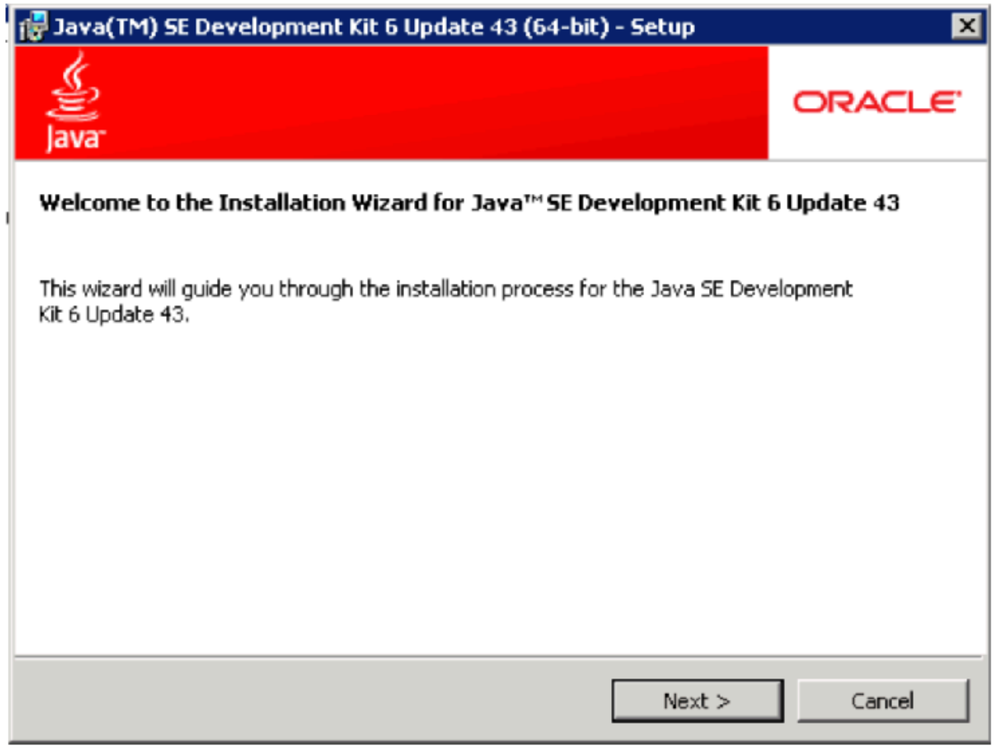
Installing the Sun Java JDK

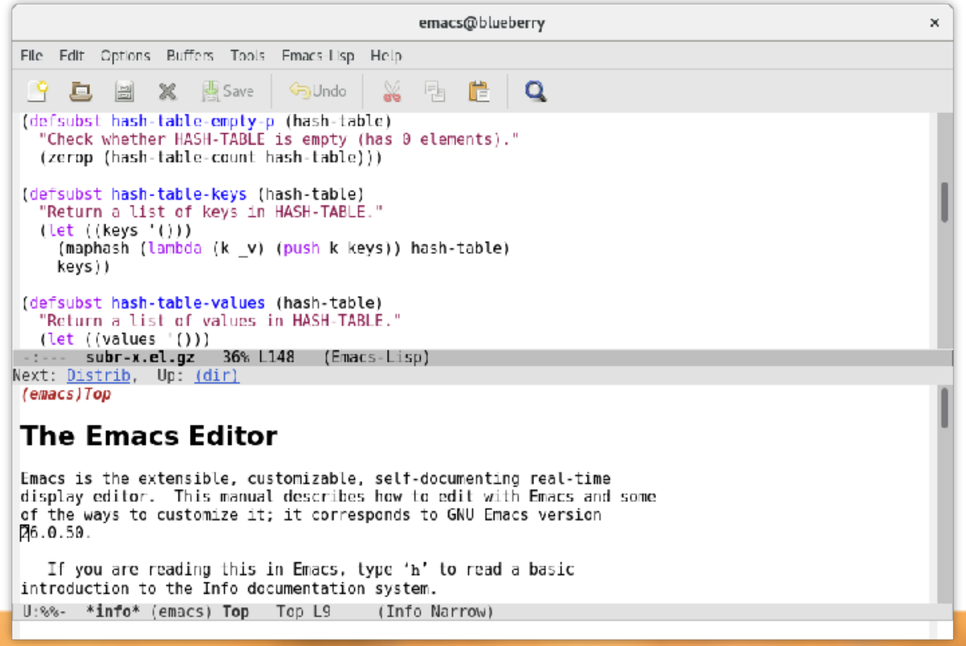
When you install the Sun Java SDK, you must install it to a path that does not have any spaces.

To install the Sun Java SDK:

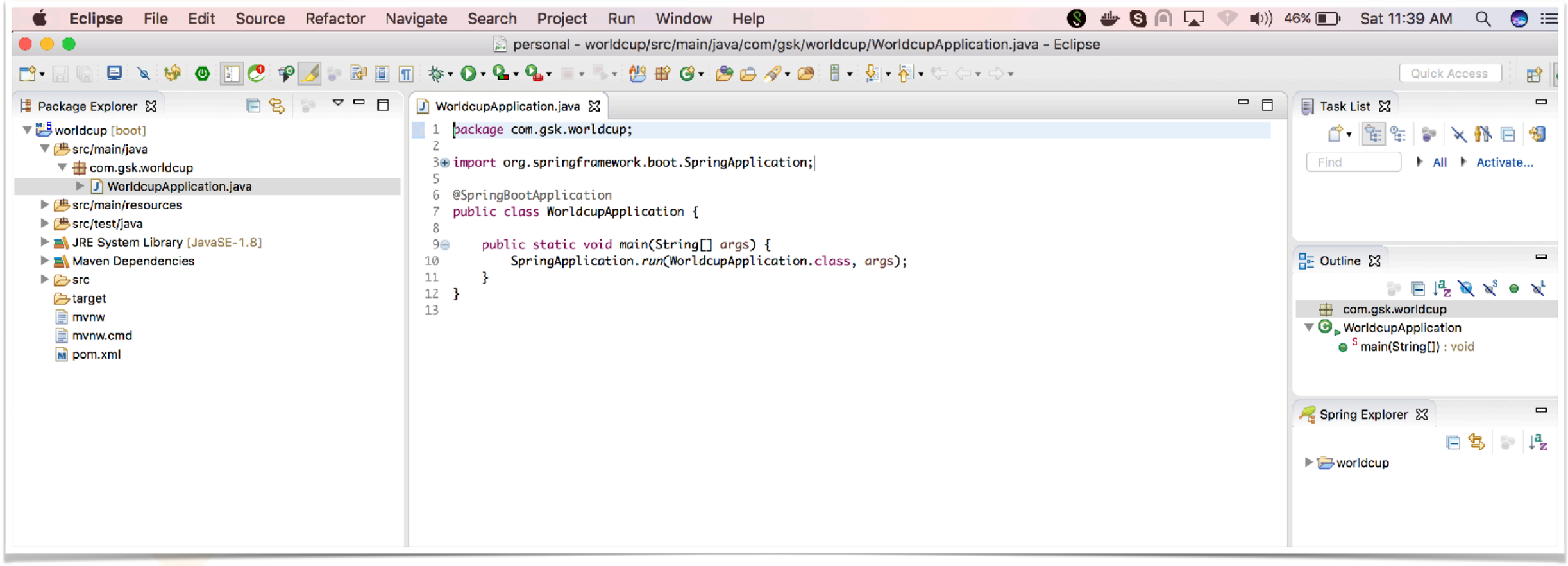
1. Double-click the JDK installer file that you downloaded (`jdk-6u43-windows-x64.exe`).

   The welcome page of the JDK installation wizard is displayed.

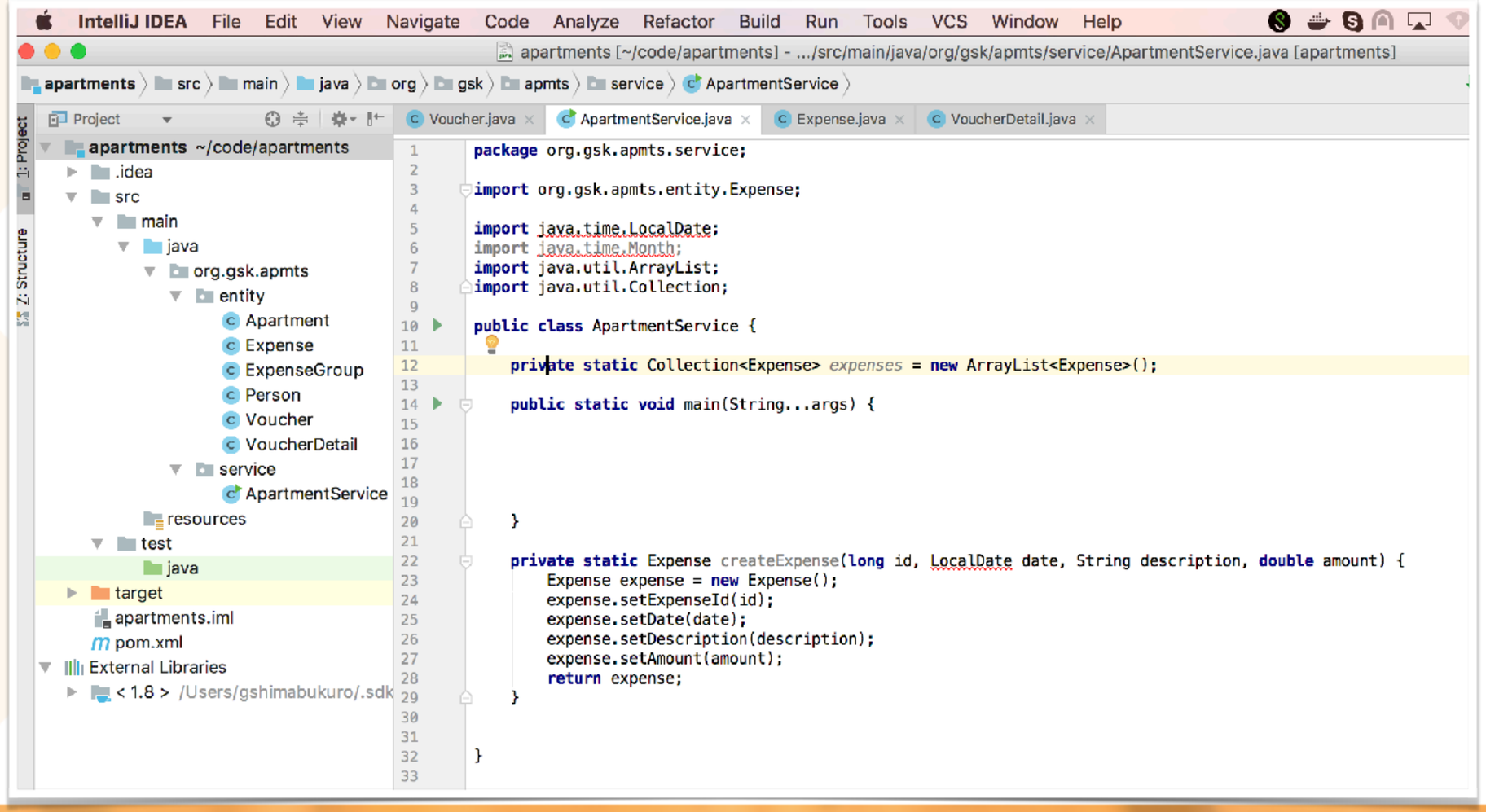




Eclipse


Eclipse File Edit Source Refactor Navigate Search Project Run Window Help
46% Sat 11:39 AM
personal - worldcup/src/main/java/com/gsk/worldcup/WorldcupApplication.java - Eclipse
Quick Access
Package Explorer
worldcup [boot]
src/main/java
com.gsk.worldcup
WorldcupApplication.java
src/main/resources
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
mvnw
mvnw.cmd
pom.xml
WorldcupApplication.java
1 package com.gsk.worldcup;
2
3 import org.springframework.boot.SpringApplication;
5
6 @SpringBootApplication
7 public class WorldcupApplication {
8
9 public static void main(String[] args) {
10 SpringApplication.run(WorldcupApplication.class, args);
11 }
12 }
13
Task List
Find
All
Activate...
Outline
com.gsk.worldcup
WorldcupApplication
main(String[]) : void
Spring Explorer
worldcup

Intellj


IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
apartments [~/code/apartments] - .../src/main/java/org/gsk/apmts/service/ApartmentService.java [apartments]
apartments src main java org gsk apmts service ApartmentService
Project
apartments ~/code/apartments
.idea
src
main
java
org.gsk.apmts
entity
Apartment
Expense
ExpenseGroup
Person
Voucher
VoucherDetail
service
ApartmentService
resources
test
java
target
apartments.iml
pom.xml
External Libraries
< 1.8 > /Users/gshimabukuro/.sdk
Voucher.java
ApartmentService.java
Expense.java
VoucherDetail.java
1: Project
7: Structure
package org.gsk.apmts.service;
import org.gsk.apmts.entity.Expense;
import java.time.LocalDate;
import java.time.Month;
import java.util.ArrayList;
import java.util.Collection;
public class ApartmentService {
private static Collection<Expense> expenses = new ArrayList<Expense>();
public static void main(String...args) {
}
private static Expense createExpense(long id, LocalDate date, String description, double amount) {
Expense expense = new Expense();
expense.setExpenseId(id);
expense.setDate(date);
expense.setDescription(description);
expense.setAmount(amount);
return expense;
}
}

Programming on Java

Data types

- Java is **strongly-typed** language.
  - Every variable has a type, every expression has a type and every type is strictly defined.
  - All assignments are checked for type compatibility.
- Defines 8 **primitive** types of data.

Data types - Primitives

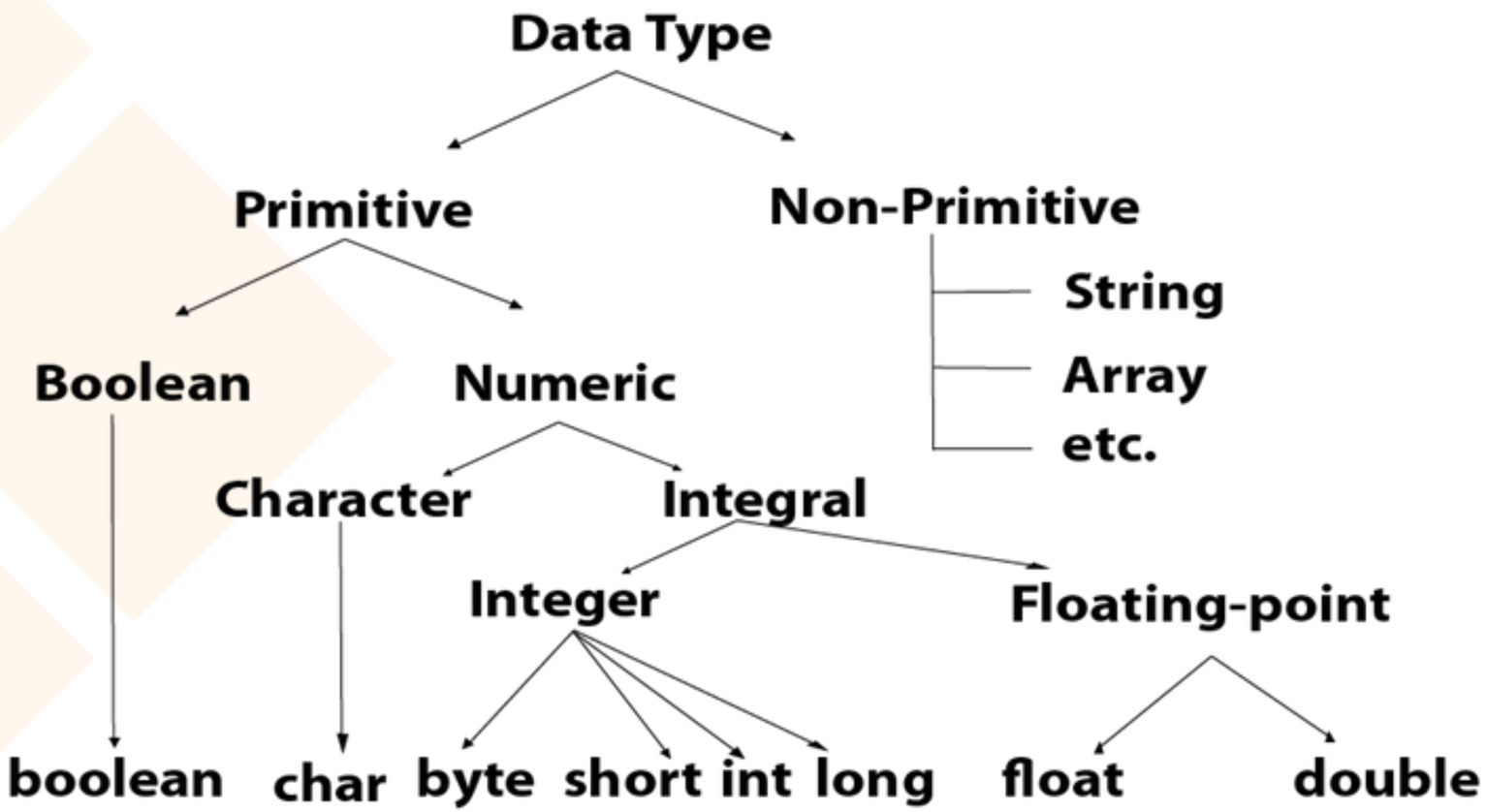
- Integers: byte, short, int and long.
- Floating-point numbers: float and double.
- Characters: char
- Boolean: boolean

Data types - Primitives

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

Data types


Data Type
Primitive
Non-Primitive
String
Array
etc.
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

Data types - Literals

- Integers: By default, int.
  - byte and short, only if is within the range.
  - long, adding an upper/lower case ‘L’ character at the end of literal value.
- Floating-point: By default, double
  - float, adding an upper/lower case ‘F’ character at the end of literal value.
- Characters: Indices into Unicode set.
- Boolean: true / false

Variables

- Basic unit of storage in a Java program.
- Defined as combination of, an identifier, a type, and optionally, an initializer.
- Scope: Determines the lifetime and visibility. In Java, there are 2 major scopes, Class and Method.

Variables - Examples

```
int a, b, c;                // declares three ints, a, b, and c.
int d = 3, e, f = 5;        // declares three more ints, initializing
                            // d and f.
byte z = 22;                // initializes z.
double pi = 3.14159;        // declares an approximation of pi.
char x = 'x';               // the variable x has the value 'x'.
```

Arrays

- Group of like-typed variables that are referred to by a common name.
- Each element is accessed by its index.
- Allows single and multiple dimensions.
- Fixed sized.

Operators

- Can be classified into 4 groups: Arithmetic, Bitwise, Relational and Boolean-logic.
- Arithmetic: Basic arithmetic operations (‘+’, ‘-’, ‘*’, ‘/’); Modulus (‘%’); Arithmetic compound (‘+=’, ‘%=’), Increment/Decrement (‘x++’, ‘++x’, ’x--’).
- Bitwise: Bitwise logical operators (‘&’, ‘I’, ‘~’, ‘^’), Left shift, right shift, Bitwise operator compound assignments, Relational operator (‘==’, ‘!=’, ‘<’, ‘>’, ‘<=’, ‘>=’); Boolean logical (just for boolean values), Short circuit (‘&&’, ‘II’)
- Assignment (‘=’)
- Simplified IF: ‘?’. (More details on conditionals)

Classes

- Declaring a class, implies its form and nature. (data and behavior)
- Defines a new type.
- Declared by using the word '***class***'.
- Class members: Methods, variables. Variable declared in a class are called "*instance variables*".
- ***main()*** method isn't necessary unless that class is the starting point of your program.

Class definition

```java
package org.gsk.apmts.service;

import org.gsk.apmts.entity.Expense;

import java.time.LocalDate;
import java.time.Month;
import java.util.ArrayList;
import java.util.Collection;

public class ApartmentService {

    private static Collection<Expense> expenses = new ArrayList<Expense>();

    public static void main(String...args) {

    }

    private static Expense createExpense(long id, LocalDate date, String description, double amount) {
        Expense expense = new Expense();
        expense.setExpenseId(id);
        expense.setDate(date);
        expense.setDescription(description);
        expense.setAmount(amount);
        return expense;
    }


}
```

Declaring objects

- Creating an object is a 2-step process.
  - Declaring a variable of the object type.
  - Acquiring a physical copy of the object and assign it to the variable.

```
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```

Methods

- Allow to define logic that will be executed.
- It could define a return-type, if the result of execution returns a value. If it does not return any value, return type must be “**void**”,
- It also could define a parameter or parameters that will be passed as input for the method.

```
type name(parameter-list) {
    // body of method
}
```

Enums

- List of named constants that define a new type and its accepted values.
- Allow to declare a set of discrete data values.
- Used for error codes, different states a device could have, and so on.

```
// An enumeration of apple varieties.
enum Apple {
  Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
```

Object-Oriented Programming (OOP)

Object - Oriented Programming

- Two paradigms: Code and Data.
- Process-oriented model: “code acting on data”. Example: C.
- Object-oriented model: “data controlling access to code”.

Abstraction

- It is a way to manage complexity of things.
- Hierarchy allows to break a complex system into a set of more manageable pieces (subsystems)
- Applied to computer programs: Data can be transformed into components objects.
- Process could be modeled as the interaction between those components. That is the essence of OOP.

The three OOP Principles

- Encapsulation
- Inheritance
- Polymorphism

Encapsulation

- Mechanism that binds together code and the data it manipulates.
- Protects the data to be arbitrarily accessed by code defined outside.
- Access to code and data is controlled through a well-defined interface.

Inheritance

- Process by which one acquires properties of another object.
- Supports the hierarchical classification concept.
- Interacts with encapsulation since any subclass of an encapsulated type, will have the same attributes plus some others, allowing that complexity grows gradually.

Polymorphism

- One interface, multiple methods.
- Having a base definition of an method, its logic could be defined or redefined by the subclasses that inherit from a base class.
- Allows to write clean, resilient, readable code.

Programming on Java - Control structures

Control Statements

- A program uses control statements to determine the flow of execution according specific conditions.
- On Java, control statements could be classified in:
  - Selection
  - Iteration and
  - Jump.

Selection

- Allows to choose different paths of execution
- Java supports two selection statements: `if` and `switch`.
- Statement “if” works like this: If the condition is true, then statement1 is executed. Otherwise, statement2 (if it exists) is executed. In no case will both statements be executed.

```
if (condition) statement1;
else statement2;
```

Selection

- Nested Ifs: One of the statement, consists or includes a new if statement with a different condition.
- “Switch” statement works like this: The value of the expression is compared with each of the values in the case statements. If a match is found, the code sequence following that case statement is executed. If none of the constants matches the value of the expression, then the default statement is executed. However, default statement is optional. If no case matches and no default is present, then no further action is taken.

```
switch (expression) {
  case value1:
     // statement sequence
     break;
  case value2:
     // statement sequence
     break;
.
.
.
  case valueN :
     // statement sequence
     break;
  default:
    // default statement sequence
}
```

Iteration statements

- Java iteration statements are: `for`, `while` and `do-while`.
- Create what we know as loops. Repeatedly executes the same set of instructions until a termination condition is met.

Iteration - While

- Java’s most fundamental loop statement.
- It repeats a statement or block while its controlling expression is true.

```
while(condition) {
    // body of loop
}
```

Iteration - Do While

- The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.

```
do {
  // body of loop
} while (condition);
```

Iteration - For

- It defines an initialization, a condition, and the way each iteration will be done.
- The for loop operates as follows. When the loop first starts, the initialization portion of the loop is executed. Next, condition is evaluated. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates. Next, the iteration portion of the loop is executed. This process repeats until the controlling expression is false.

```
for(initialization; condition; iteration) {
  // body
}
```

Iteration - For each

- Newer form of “for” statement.
- It’s designed to cycle through a collection of elements, such as an array.
- A “pivot” object is defined with the same type of the collection elements, to take the values of each of them, one by one, during the execution of the statement.

for(*type itr-var* : *collection*) *statement-block*

Jump statements

- These statements transfer control to another part of your program.
- Java Jump stamens are: `break`, `continue` and `return`.

Jump - break

- It can be used in 3 ways.
  - To terminate a statement sequence on a switch statement.
  - To exit a loop or iteration, bypassing conditional expression and remaining code in the body of the loop.
  - As a “civilized” form of goto, also known as labeled break statement.

Jump - break - goto

```
// Using break as a civilized form of goto.
class Break {
  public static void main(String args[]) {
    boolean t = true;

    first: {
      second: {
        third: {
          System.out.println("Before the break.");
          if(t) break second; // break out of second block
          System.out.println("This won't execute");
        }
        System.out.println("This won't execute");
      }
      System.out.println("This is after second block.");
    }
  }
}
```

Jump - continue

- It forces an early iteration of a loop.
- Loop will keep running but the remaining code for executed iteration don’t.
- Like break statement, it allows labels to determine which enclosing loop to continue.

Jump - return

- It is used to explicitly return from a method. That cause the control to be transfer back to method caller.
- It immediately terminates the execution of the method in which is executed.
- All code after return statement will be consider as unreachable by the compiler. “If” statements works fine to prevent this kind of errors.

Java Virtual Machine (JVM)

Java Virtual Machine (JVM)

- Cornerstone of Java platform.
- Abstract computing machine, it has an instruction set and manipulates memory areas at runtime.
- Compiled to be executed by JVM is represented using Class file format (.class).

JVM Runtime Areas

- pc (Program counter): For each thread, it contains the address of JVM instruction currently being executed.
- Stack: By each thread, hold local variables and partial results. May be heap allocated.
- Heap: Shared by all threads. All class instances and arrays are allocated there. Heap storage for object is reclaimed by Garbage Collector. When there isn't available storage, JVM throws an `OutOfMemoryError`.

Java Runtime Areas

- Method area: Shared, logically part of Heap but could be not available for GC reallocation depending on JVM implementation.
    - Run-Time Constant Pool: Symbol table. Constructed when class or interface is created by JVM
    - Native method stack

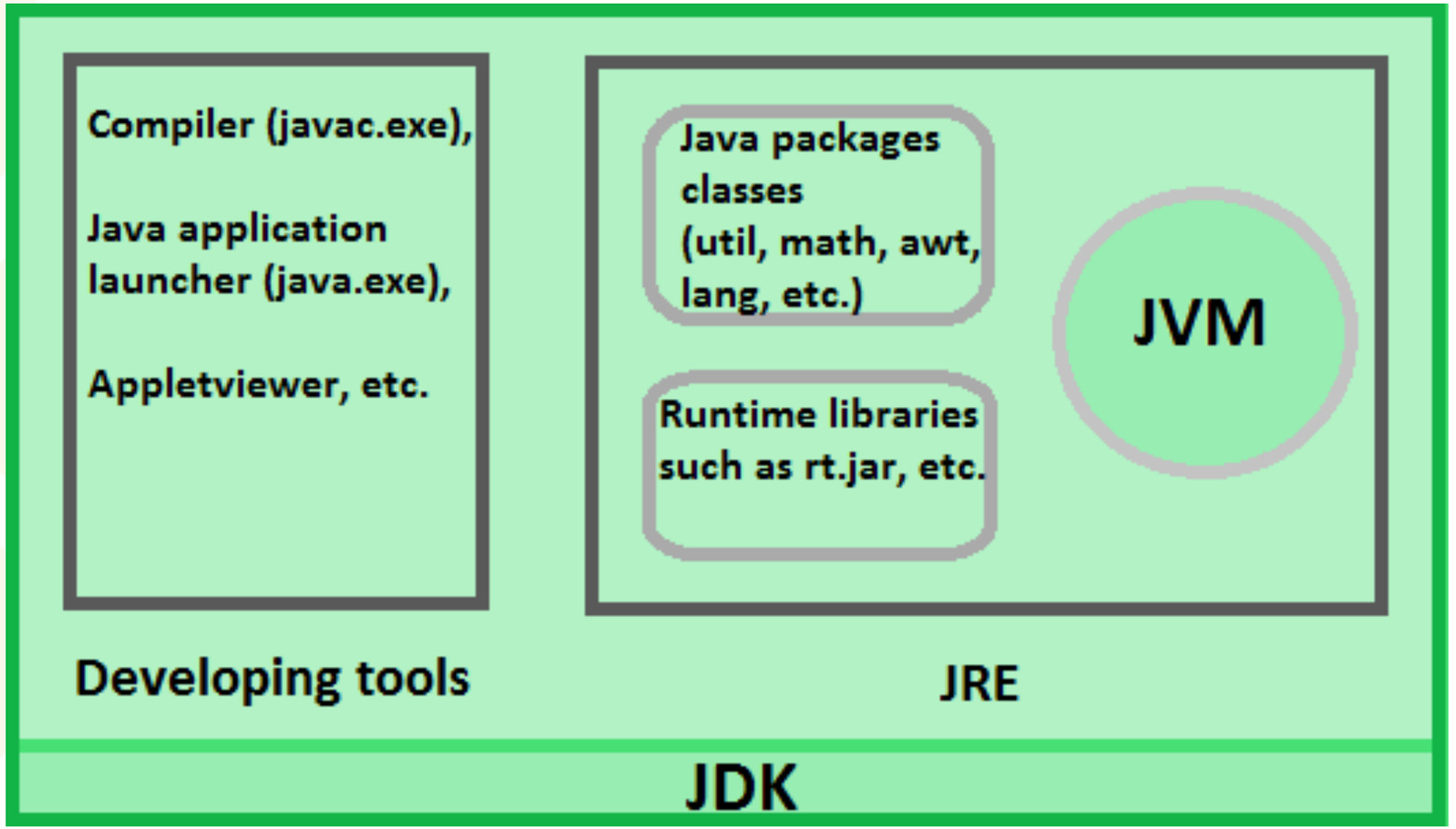
Java Run-Time Environment (JRE)

- Software layer running on top of device's operating system that allows to execute applications written on Java and Java-based languages.
- Contains the JVM, and additionally the Class Loader and the Java class libraries.

Java Development Kit (JDK)

- Bundles JRE.
- Also includes the Java Compiler (javac), an interpreter/loader (java), archiver (jar), document generator (javadoc) and other tools like debugging.
- GUI tools: JConsole, Java VisualVM and Oracle Mission Control.

Difference between JVM, JRE and JDK



Programming on Java: Exceptions

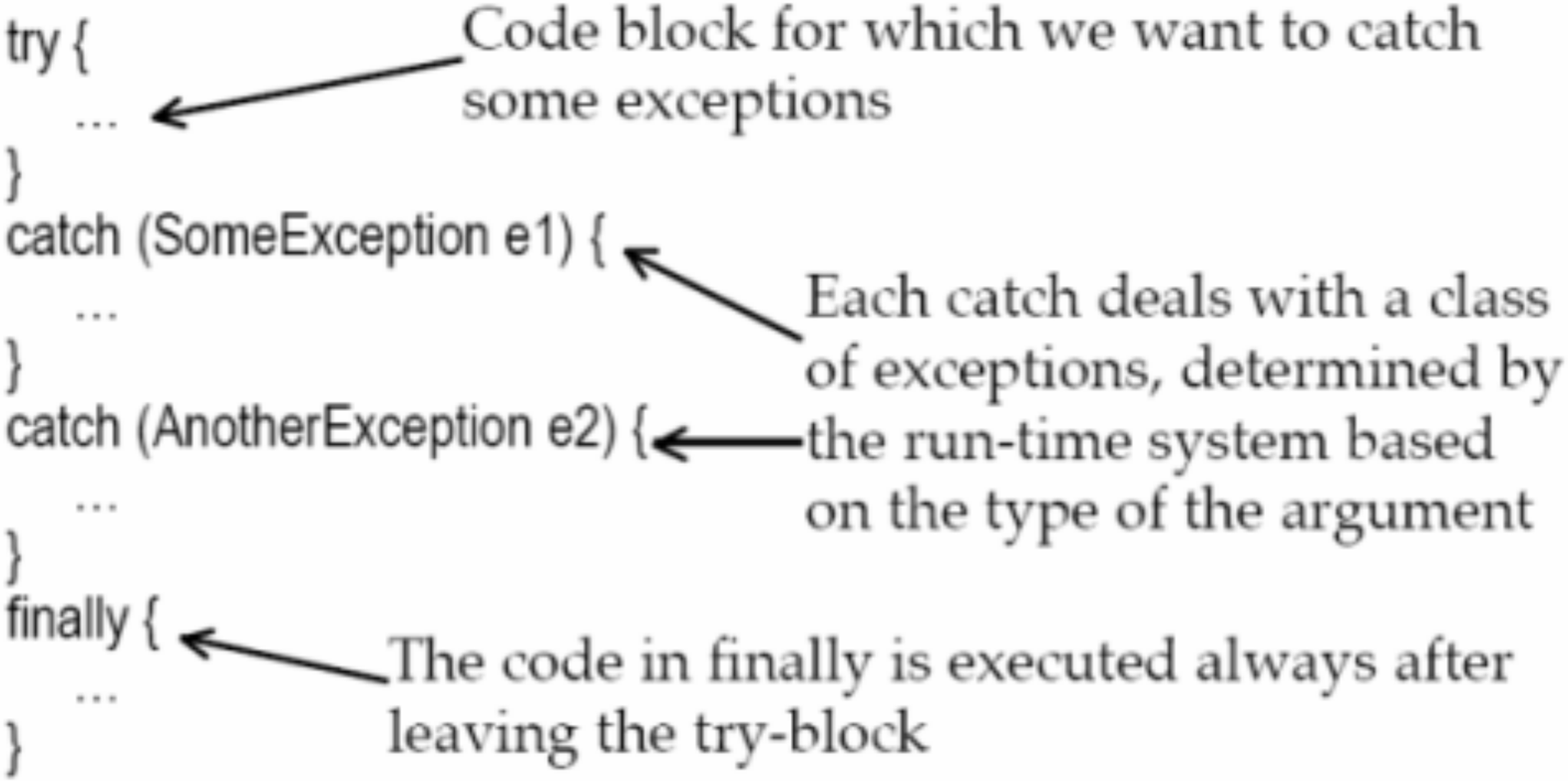
Exceptions

- Allows to handle unexpected behavior during run-time.
- Exceptions are objects, and all exception objects class should extends `Throwable` class.
- Could be classified on: Checked and Unchecked exceptions.

Exceptions

- throw: Expects an throwable value. It raises the exception interrupting the execution of the method on which is executed.
- throws: It transfers the control of the exception to the caller of the method on which is executed.

Exceptions


try {
...
}
catch (SomeException e1) {
...
}
catch (AnotherException e2) {
...
}
finally {
...
}
Code block for which we want to catch some exceptions
Each catch deals with a class of exceptions, determined by the run-time system based on the type of the argument
The code in finally is executed always after leaving the try-block