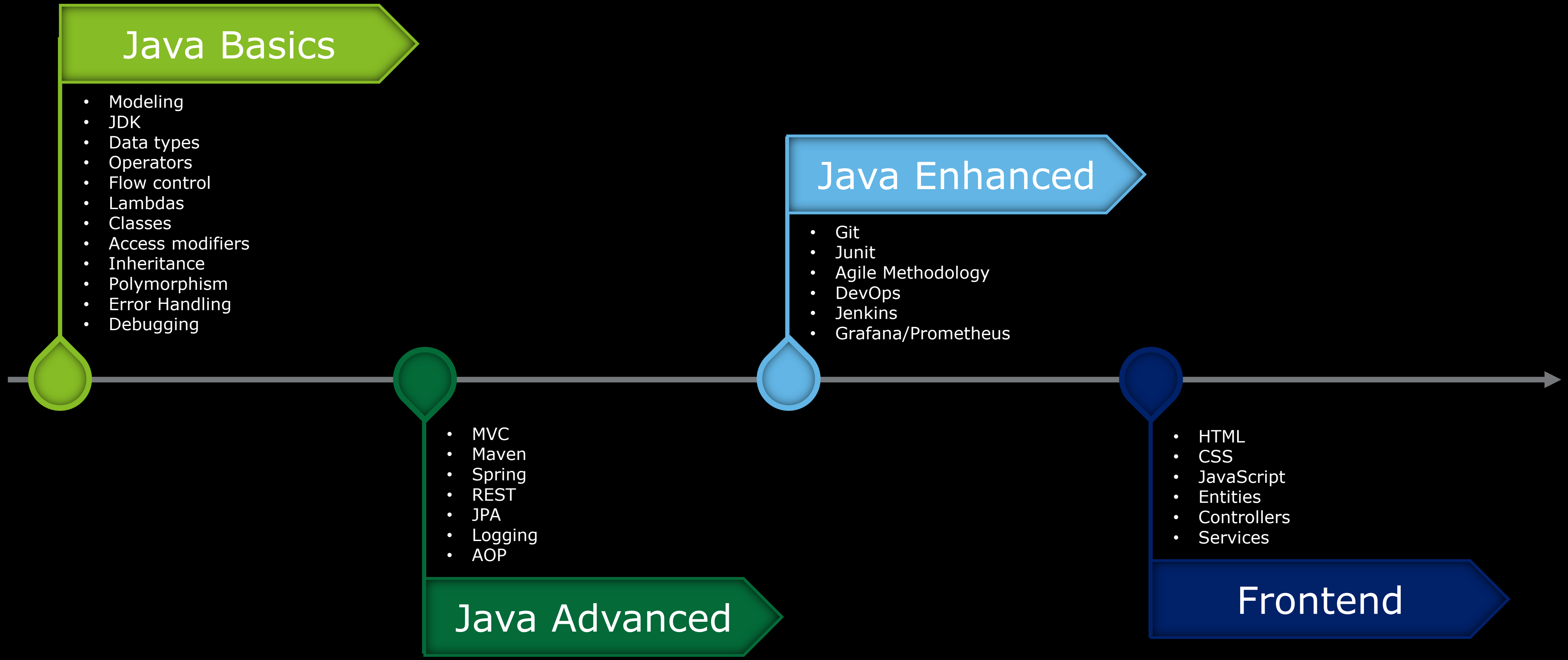
Deloitte.

Java Essentials

Java Academy

Deloitte, September 2019

Agenda
Detailed



Java Basics

- Modeling
- JDK
- Data types
- Operators
- Flow control
- Lambdas
- Classes
- Access modifiers
- Inheritance
- Polymorphism
- Error Handling
- Debugging

Java Advanced

- MVC
- Maven
- Spring
- REST
- JPA
- Logging
- AOP

Java Enhanced

- Git
- Junit
- Agile Methodology
- DevOps
- Jenkins
- Grafana/Prometheus

Frontend

- HTML
- CSS
- JavaScript
- Entities
- Controllers
- Services

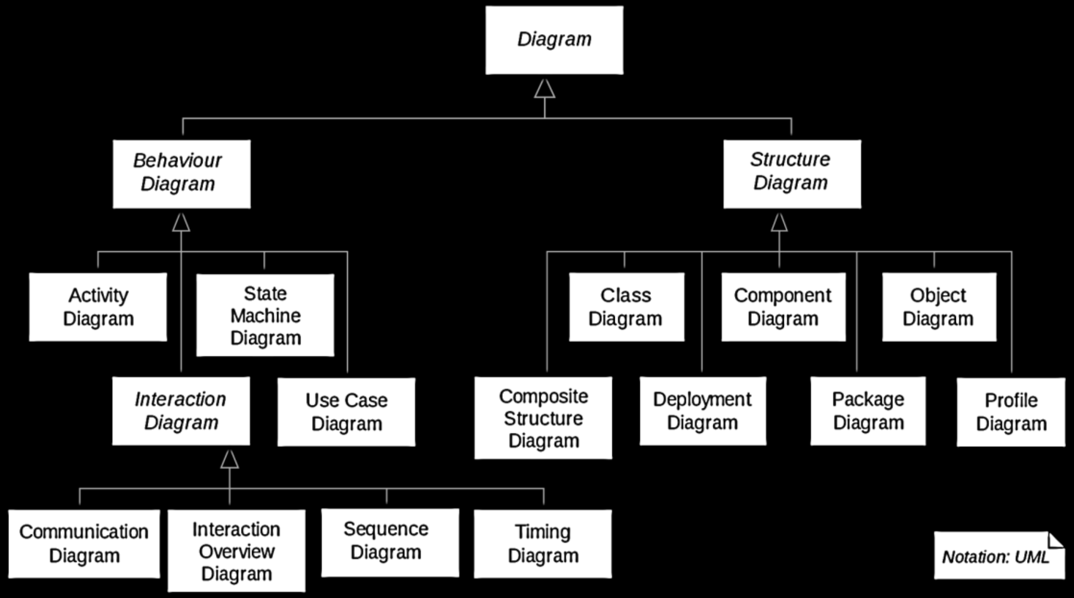
Day 1

Java Basics

Modeling

Java Basics
Modeling

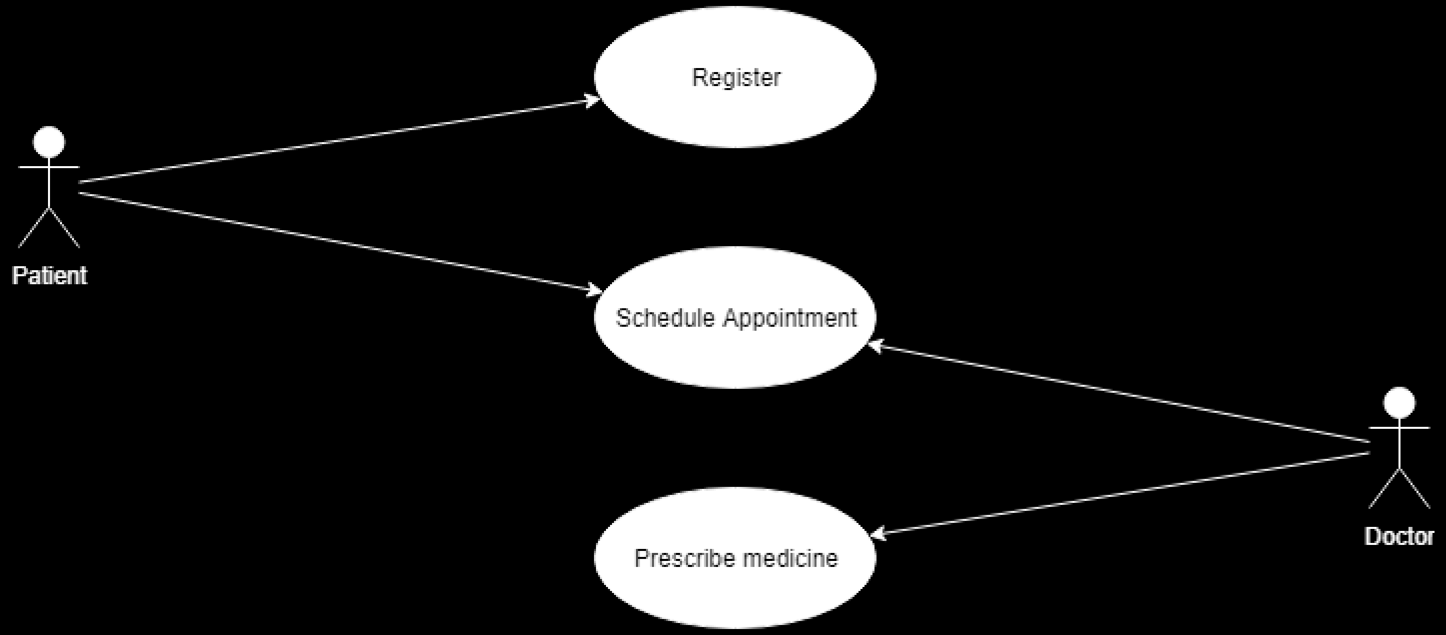
- UML – Unified Modeling Language
- Created around 1990s
- Latest version (2.5.1) released in 2017
- Provides a standard way to visualize the design of a system



Java Basics

Modeling - Examples

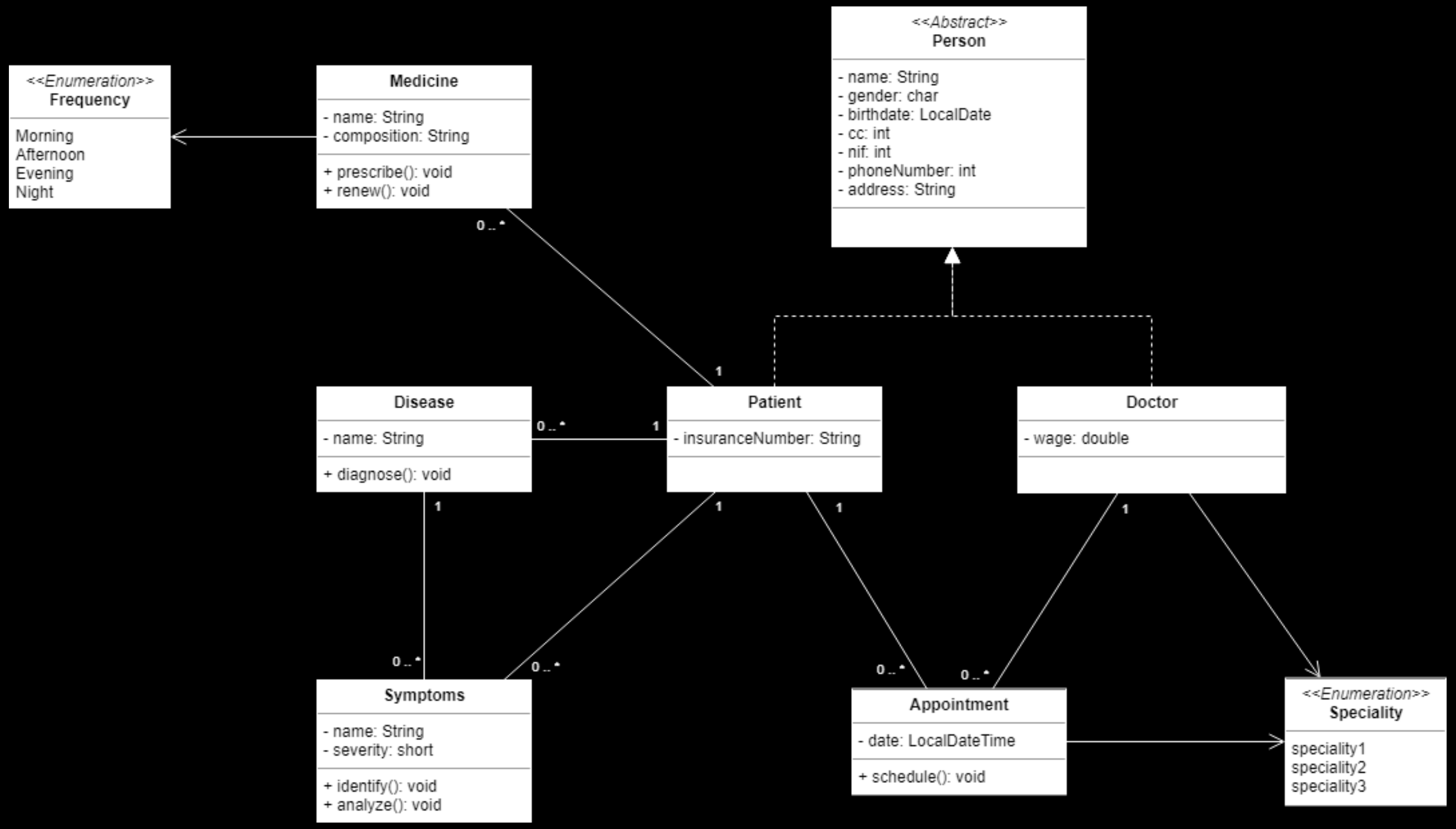
Use case diagram



Java Basics

Modeling - Examples

Class diagram



<<Enumeration>>
Frequency

Morning
Afternoon
Evening
Night

Medicine

- name: String
- composition: String

+ prescribe(): void
+ renew(): void

<<Abstract>>
Person

- name: String
- gender: char
- birthdate: LocalDate
- cc: int
- nif: int
- phoneNumber: int
- address: String

Disease

- name: String

+ diagnose(): void

Patient

- insuranceNumber: String

Doctor

- wage: double

Symptoms

- name: String
- severity: short

+ identify(): void
+ analyze(): void

Appointment

- date: LocalDateTime

+ schedule(): void

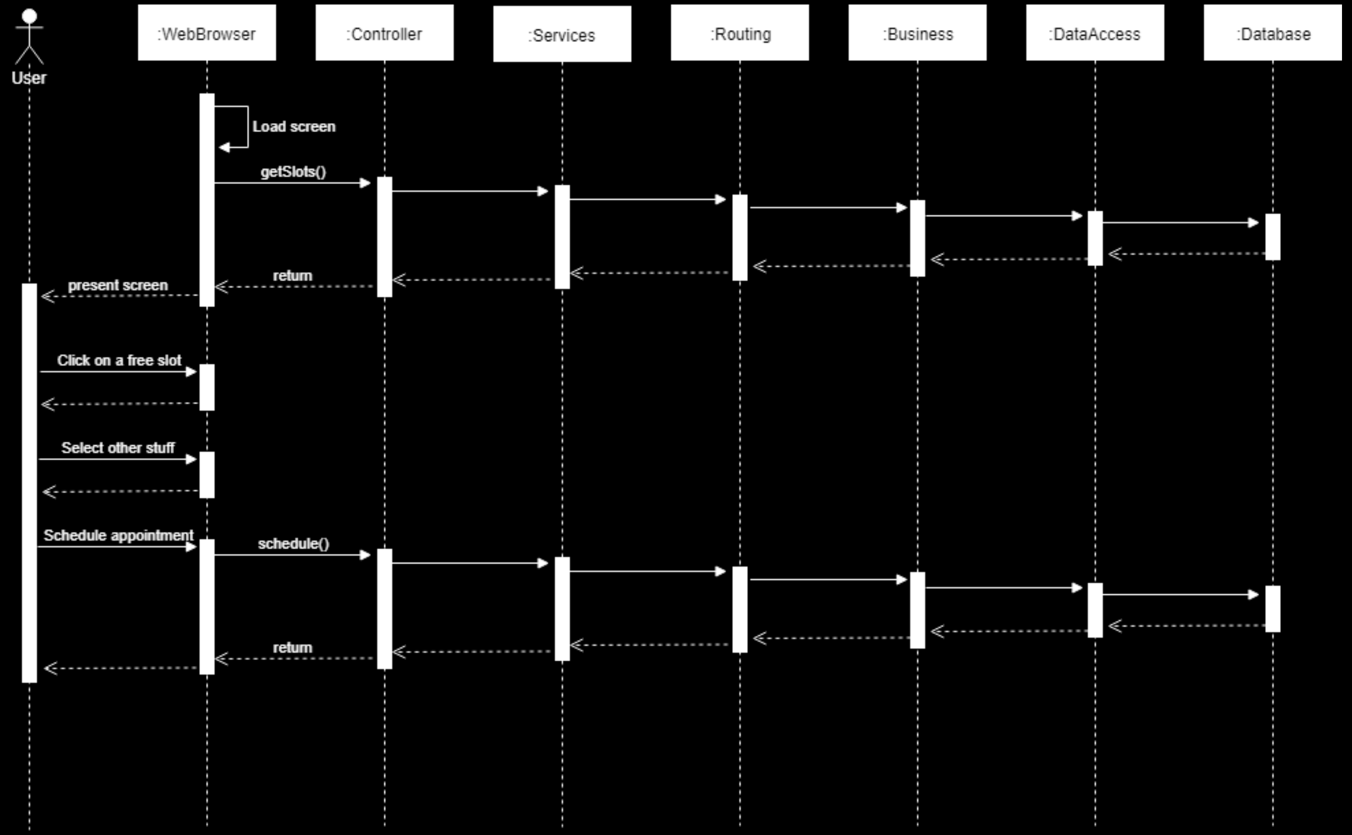
<<Enumeration>>
Speciality

speciality1
speciality2
speciality3

Java Basics

Modeling - Examples

Use case diagram



Java

Java Basics

Java

- Developed in 1990s by Sun Microsystems
- Agnostic to the environment
- Object oriented programing language (OOP)
- Main advantages:
  - Modularity
  - Information hiding
  - Code reuse
  - Maintainability

Java Basics

JDK

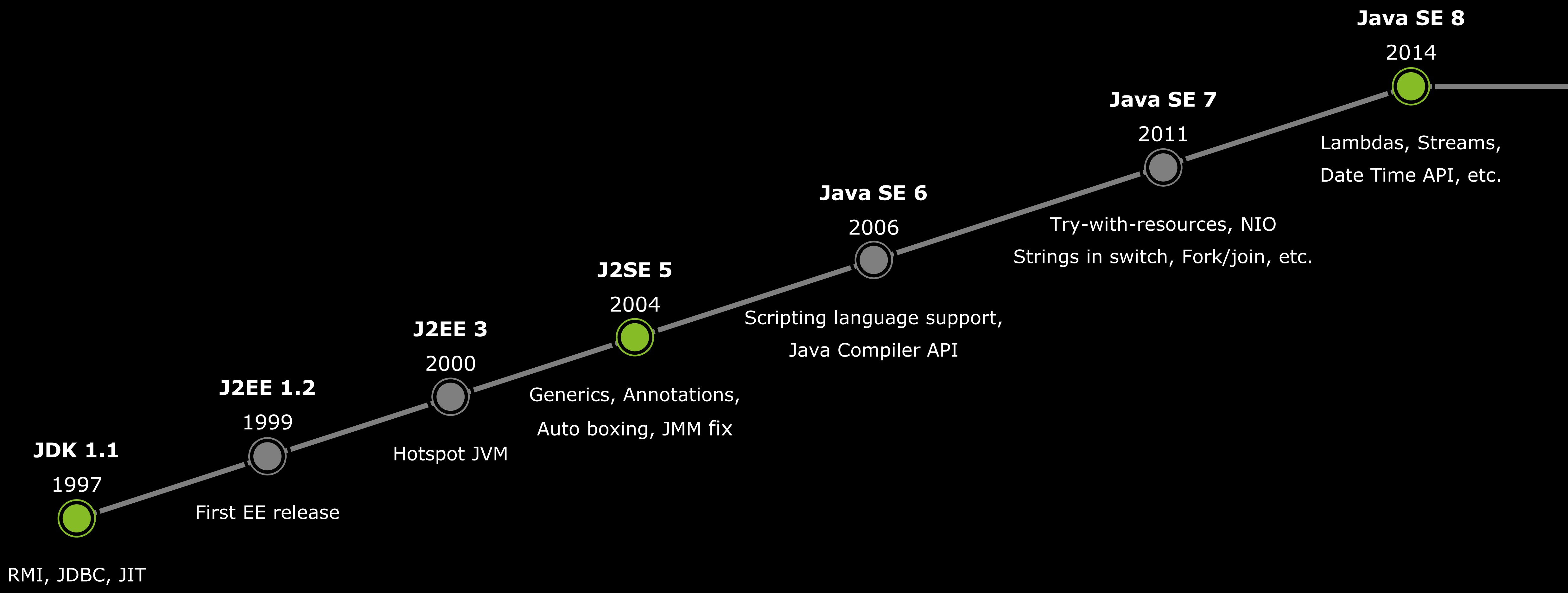
- Combined, the JVM software and Java class libraries are referred to as the Java Runtime Environment (JRE). Java Runtime Environments are available from Oracle for many common platforms

- JDK includes the JRE plus development tools (compilers and debuggers) that are necessary or useful for developing Java applications

- There is an open source version of JDK - OpenJDK

Java Basics

Version Release Roadmap



JDK 1.1
1997
RMI, JDBC, JIT

J2EE 1.2
1999
First EE release

J2EE 3
2000
Hotspot JVM

J2SE 5
2004
Generics, Annotations,
Auto boxing, JMM fix

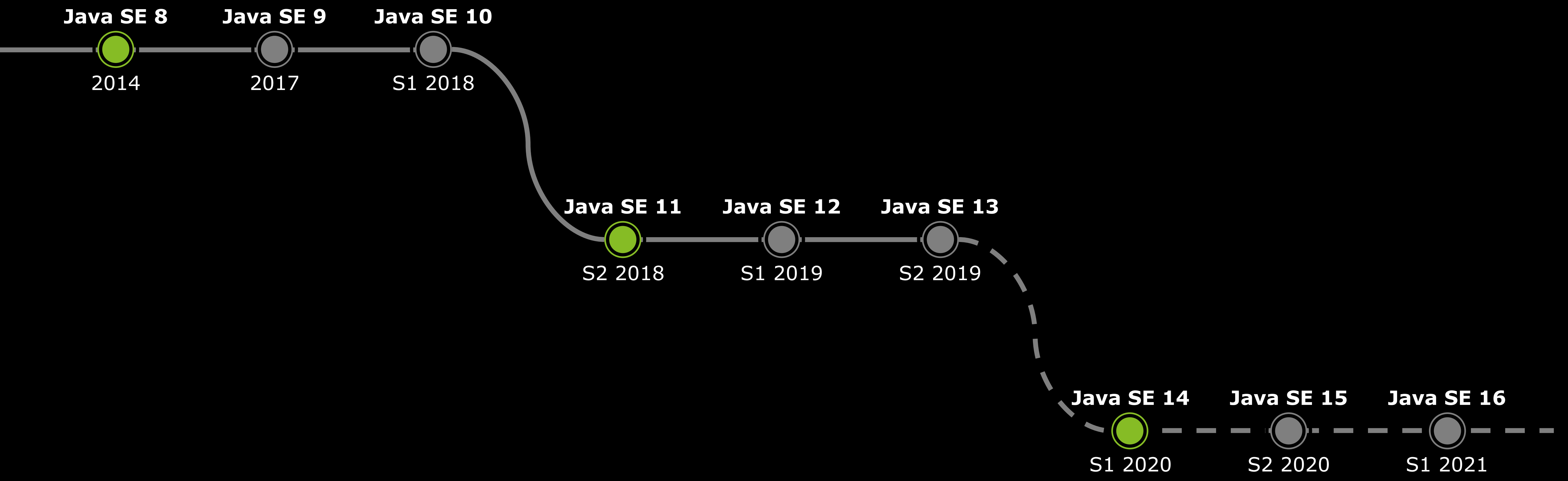
Java SE 6
2006
Scripting language support,
Java Compiler API

Java SE 7
2011
Try-with-resources, NIO
Strings in switch, Fork/join, etc.

Java SE 8
2014
Lambdas, Streams,
Date Time API, etc.

Java Basics

Version Release Roadmap



Java SE 8
2014

Java SE 9
2017

Java SE 10
S1 2018

Java SE 11
S2 2018

Java SE 12
S1 2019

Java SE 13
S2 2019

Java SE 14
S1 2020

Java SE 15
S2 2020

Java SE 16
S1 2021

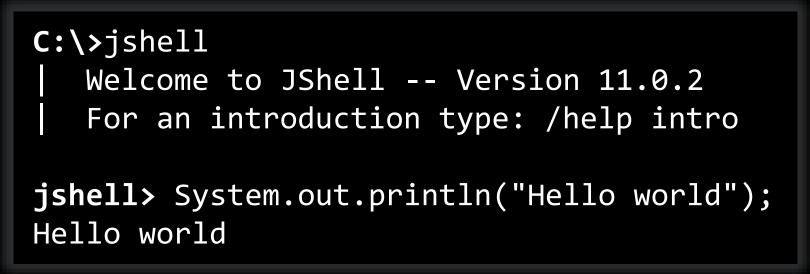
Java Basics

JShel

- Released in Java SE 9

- Read-Eval-Print loop

- Interactive computer programming environment



```
C:\>jshell
|  Welcome to JShell -- Version 11.0.2
|  For an introduction type: /help intro

jshell> System.out.println("Hello world");
Hello world
```

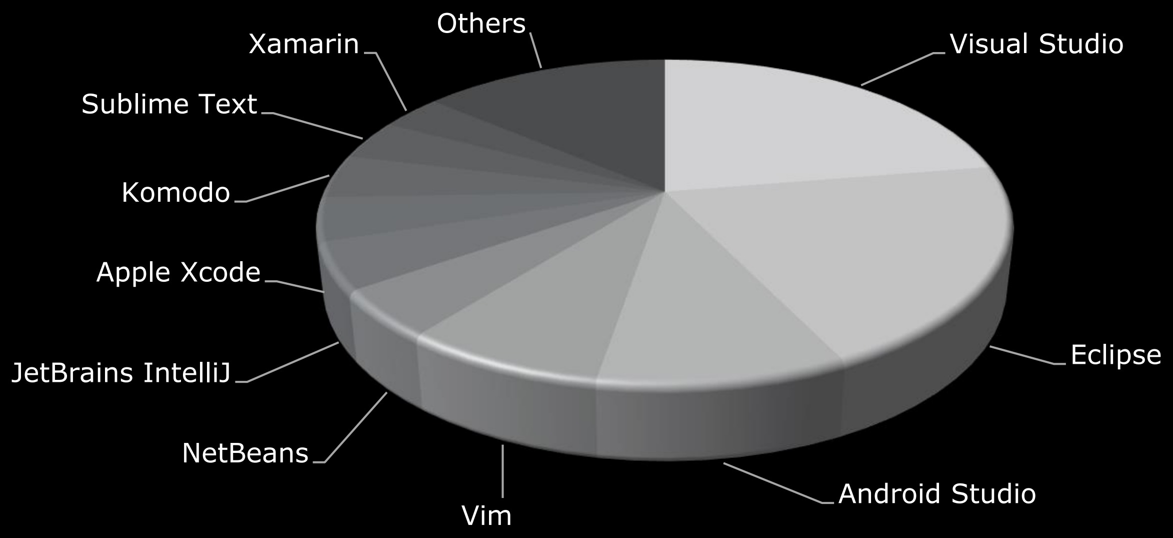
1. Takes single user inputs

2. Evaluates (executes) them

3. Returns the result to the user

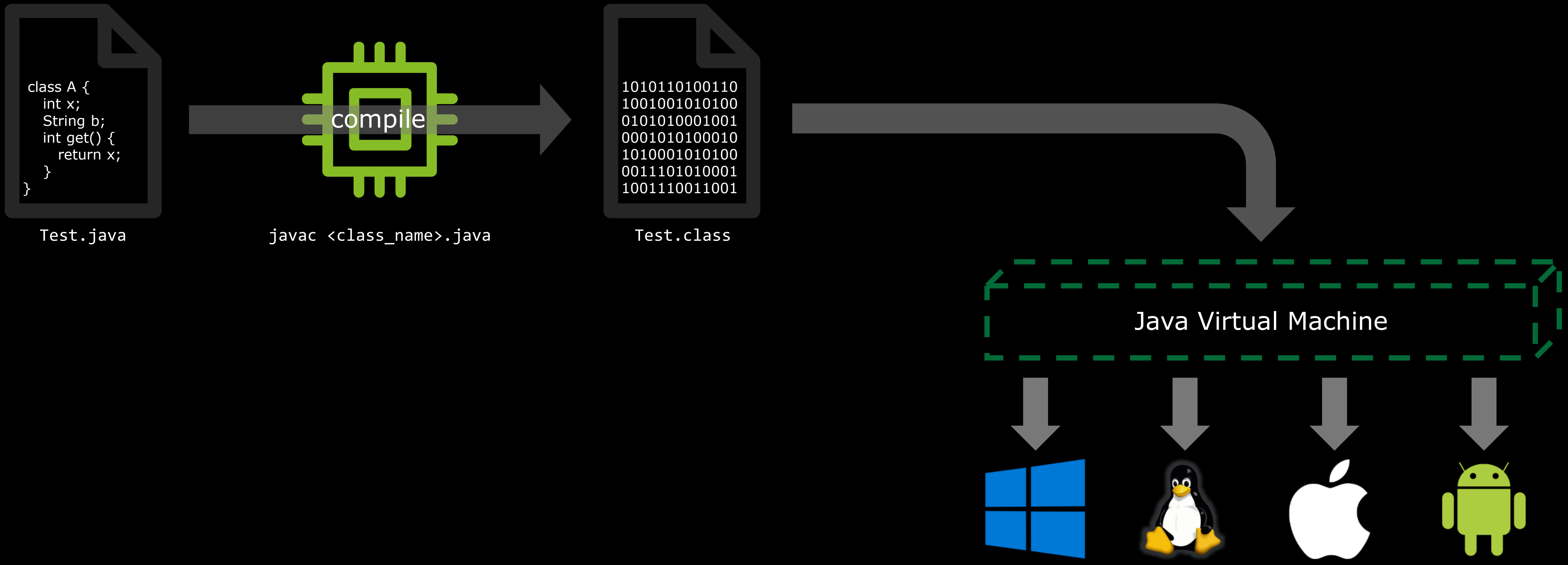
Java Basics
IDE

- Integrated development environment
  - Source code editor
  - Compiler/interpreter
  - Build automation tools
  - Debugger



Java Basics

Compiling and running a Java program



```
class A {
   int x;
   String b;
   int get() {
      return x;
   }
}
```

```
javac <class_name>.java
```

```
1010110100110
1001001010100
0101010001001
0001010100010
1010001010100
0011101010001
1001110011001
```

Java Basics
Class

- Represents a complex data type
- “Blueprint" for creating objects
- Are composed by:
  - **Attributes:** types of data that make up the entity (what they can store)
  - **Methods:** procedures that the entity can perform (what they can do)

```java
import java.time.LocalDate;

public class Medicine {

    private int id;
    private String name;
    private String composition;
    private LocalDate expirationDate;
    private Frequency frequency;
    private int quantity;

    public Medicine(int id, String name, String composition, LocalDate expirationDate) {
        this.id = id;
        this.name = name;
        this.composition = composition;
        this.expirationDate = expirationDate;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }


    //...

}
```

Java Basics

Method

- Used to perform certain actions, and are also known as functions

- Block of code which only runs when it is called

- You can pass data, known as parameters, and can return any type of result

- Why use methods?
  - Reutilization: define the code once, and use it many times
  - Organization: segment the functionalities in smaller modules

```
public Medicine(int id, String name, String composition, LocalDate expirationDate) {
    this.id = id;
    this.name = name;
    this.composition = composition;
    this.expirationDate = expirationDate;
    this.frequency = frequency;
    this.quantity = quantity;
}
```

```
public boolean checkAvailability(LocalDateTime appointmentDate) {
    List<LocalDateTime> busySlots = getBusySlots();

    for(LocalDateTime slot : busySlots) {
        if (appointmentDate.isEqual(slot)) {
            return false;
        }
    }

    return true;
}
```

Java Basics
Pass-by-value vs Pass-by-reference

- Java manipulates objects by reference, and all object variables are references
- When we pass the value of an object, we are passing the *reference* to it
- Still, Java always passes parameters **by value**.

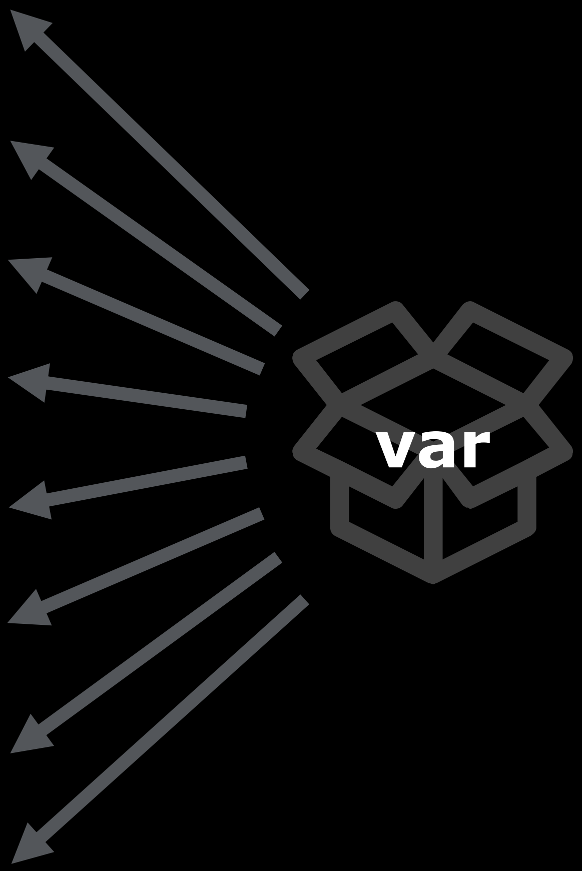
```
Person aPerson = new Person("Max");
Person backupPerson = aPerson;

// we pass the Person to the method
changeName(aPerson);
// variable is still pointing to the "Max" person when method returns
aPerson.getName().equals("Max");          // true
aPerson.getName().equals("John");         // false
aPerson == backupPerson;                  // true


public static void changeName(Person p) {
    p.getName().equals("Max");                // true
    // change "p" to point to a new Person instance "John"
    p = new Person("John");
    p.getName().equals("John");               // true
}
```

Data types

Java Basics

Data Types

byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

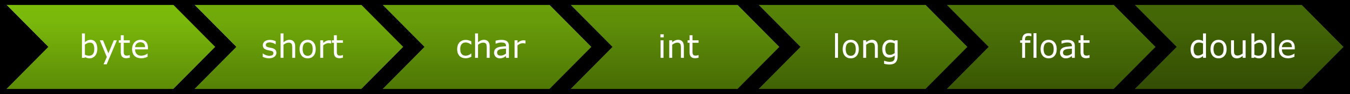


Java Basics

Data Types

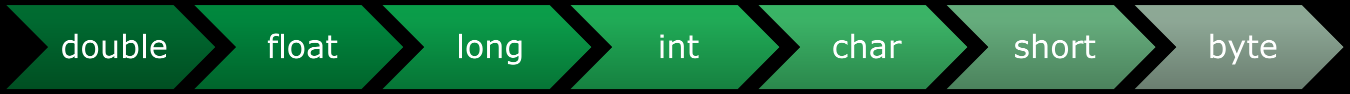
In Java, there are two types of casting:

- Widening Casting (automatically) – converting a smaller type to a larger type size



byte short char int long float double

- Narrowing Casting (manually) – converting a larger type to a smaller size type **(loses precision)**



double float long int char short byte

Java Basics

String

- An object that represents sequence of char values

Escape character	Result
\'	Single quote (')
\"	Double quote (")
\\	Backslash (\ )
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace

```
String txt = "Hello World";

txt.toUpperCase()                            // "HELLO WORLD"
txt.toLowerCase()                            // "hello world"
txt.length()                                 // 11
txt.indexOf("World")                         // 6
txt.subString(6)                             // "World"


String firstName = "John";
String lastName = "Smith";

firstName + " " + lastName                   // "John Smith"
firstName.concat(" ").concat(lastName)       // "John Smith"

firstName                                    // ???
```

Java Basics

String vs StringBuilder vs SpringBuffer

- String objects are immutable
- Mutable alternatives are StringBuffer and StringBuilder
- StringBuilder is faster
- StringBuffer is thread safe

```
String s1 = "Person";
s1.concat1("1");
System.out.println(s1);    // "Person"

StringBuilder s2 = new StringBuilder("Person");
s2.append("2");
System.out.println(s2);    // "Person2"

StringBuffer s3 = new StringBuffer("Person");
s3.append("3");
System.out.println(s3);    // "Person3"
```

Java Basics
Arrays

- Indexed container that holds a set of values of a single type
- Each item in an array is called an element
- Each element is accessed by its numerical index
- The index of the first element is 0 (zero)

```
int[] ages = { 19, 42, 92 };
ages[0];            // 19
ages[1];            // 42
ages[2];            // 92

String[] names = new String[3];
names[0] = "Mary";
names[1] = "Bob";
names[2] = "Carlos";


int[][] matrix = { { 1, 2, 3, 4 }, { 5, 6, 7 } };
matrix[0];          // { 1, 2, 3, 4 }
matrix[0][2];       // 3
matrix[1][0];       // 5
```

Java Basics
Lists

- It is an ordered collection of objects in which duplicate values can be stored
- Since List preserves the insertion order, it allows positional access and insertion of elements
- We can choose between the following List implementations in the Java Collections API:
  - java.util.ArrayList
  - java.util.LinkedList
  - java.util.Vector
  - java.util.Stack

```
List<String> list = new ArrayList<>();

list.add("one");
list.add("two");

list.get(0);           // "one"
list.indexOf("two");   // 1

list.remove(0);        // ["two"]

List.size();           // 1
```

Java Basics
Date API

- Allows us to manage dates and times

LocalDate	Represents a date (year, month, day (yyyy-MM-dd))
LocalTime	Represents a time (hour, minute, second and milliseconds (HH-mm-ss-zzz))
LocalDateTime	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss.zzz)
Instant	Represents the number of nanoseconds passed since Unix Epoch time (01-01-1970)
Duration	Amount of time modeled in terms of Instant and Time
Period	Amount of time modeled in terms of Years, Months and Days
DateTimeFormatter	Formatter for displaying and parsing date-time objects

Java Basics

Date API

```
LocalDate localDate = LocalDate.parse("2016-04-20");
LocalTime localTime = LocalTime.parse("16:30:01");

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
LocalDate parsedDate = LocalDate.parse("20-04-2016", formatter);
System.out.println(parsedDate.format(formatter));

LocalDate date = LocalDate.of(2016, 4, 20);
date = date.plusYears(2).minusDays(10);                         // date = 2018-04-10
date.withYear(2011);                                            // date = 2018-04-10
LocalDate date2 = date.plusWeeks(3);                            // date2 = 2018-05-01
LocalDate date3 = date.withYear(2011);                          // date3 = 2011-04-10

Instant minute = Instant.ofEpochSecond(3);                      // 1970-01-01T00:00:03Z
Duration duration = Duration.between(time1, time2);
Period tenDays = Period.ofDays(10);
```

Java Basics

Files

- Allows us to work with files

```
Path p = Paths.get("inexistent_file.txt");

Files.notExists(p);
Files.createFile(p);
Files.delete(p);

String write = "some text“;
Files.write(p, write.getBytes());

String read = Files.readAllLines(path).get(0);
```

Java Basics
Exercise

- License plate
- Expiration date
- Telephone
- Id
- Timestamp
- Data de Nascimento
- Cloud stored file
- File available in a web service
- Template file for download

Operators

Java Basics
Operators

- Arithmetic
- Comparison
- Logical
- Bitwise
- Assignment

Java Basics

Arithmetic operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
++	Increment	x++
--	Decrement	--x

Java Basics

Comparison operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y
instanceof	Instance of	x instanceof y

Java Basics

Logical operators

Operator	Name	Example
&& or &	And	x < 5 && x < 10
|| or |	Inclusive or	x < 5 || x < 4
^	Exclusive or	x ^ y
!	Logical not	!(x < 5 && x < 10)

Java Basics

Bitwise operators

Operator	Name	Example
<<	Left Shift	10 << 3
>>	Right Shift	20 >> 3
>>>	Right Shift (parity bit swap)	-20 >>> 3

Java Basics

Assignment operators

Operator	Name	Example
=	Assign	x = 3;

Java Basics

Assignment operators (abbreviations)

Operator	Example	Same as
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Java Basics

Math API

- Allows us to perform mathematical tasks on numbers

```
Math.max(5, 10);        // 10
Math.min(5, 10);        // 5
Math.sqrt(64);          // 8
Math.abs(-4.7);         // 4.7
Math.random();          // irrational number between 0 and 1
```

Java Basics

Exercise

- Following the rules of precedence, what is the result of **a**?

```
int a = 25 - 5 * 5 / 2 - 10 + 4;
```

- What is the type and result of **b**?

```
var b = (4 + 3 * 2 > 20 % 3 * 10) ? 10 : 20;
```

Flow control

Java Basics
Decisions

```
if (condition) {
    // block of code to be executed if condition is true
} else {
    // block of code to be executed if condition is false
}
```

```
<condition> ? <expression if true> : <expression if false>
```

Java Basics

Decisions

```
if (condition1) {
    // block of code to be executed if condition1 is true
} else if (condition2) {
    // block of code to be executed if condition1 is false and condition2 is true
} else if (condition3) {
    // block of code to be executed if condition2 is false and condition4 is true
} else {
    // block of code to be executed if condition2 and condition3 is false
}
```

Java Basics

Decisions

```
switch(expression) {
    case x:
        // block of code to be executed if expression equals to x
        break;
    case y:
        // code block
        break;
    default:
        // code block
}
```

Java Basics

Exercise

- Do an action depending on the emptiness of list
- Do an action depending on the state of element
- Do an action depending on the size of a list

Java Basics

Loops

```
while (condition) {
    // block of code to be executed
}
```

```
do {
    // block of code to be executed
} while (condition);
```

Java Basics

Loops

```
for (initialization; end condition; loop action) {
    // block of code to be executed
}
```

```
for (type variable : array_name) {
    // block of code to be executed
}
```

Java Basics

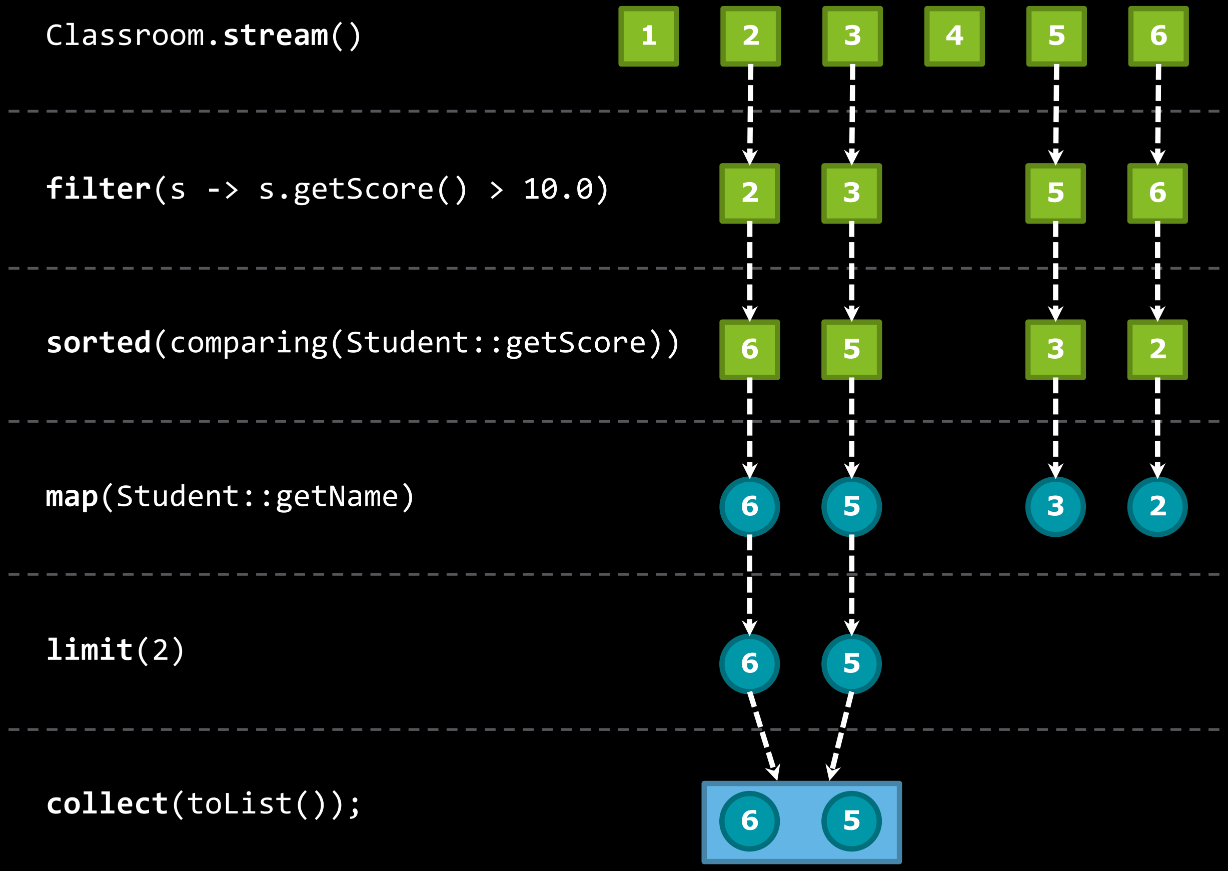
Branching statements

```java
label1: for (int a : array1) {
    label2: for (int b : array2) {
        if (a > b) {
            break;
        } else if (a + 10 > b) {
            break label1;
        } else if (a < b) {
            continue;
        } else {
            return;
        }
    }
}
```

Java Basics
Streams

- Get the names of the 2 best students in a classroom:

```
List<String> nameList = classroom.stream()
     .filter(s -> s.getScore() > 10.0)
     .sorted(comparing(Student::getScore))
     .map(Student::getName)
     .limit(2)
     .collect(toList());
```



Classroom.**stream**()

filter(s -> s.getScore() > 10.0)

sorted(comparing(Student::getScore))

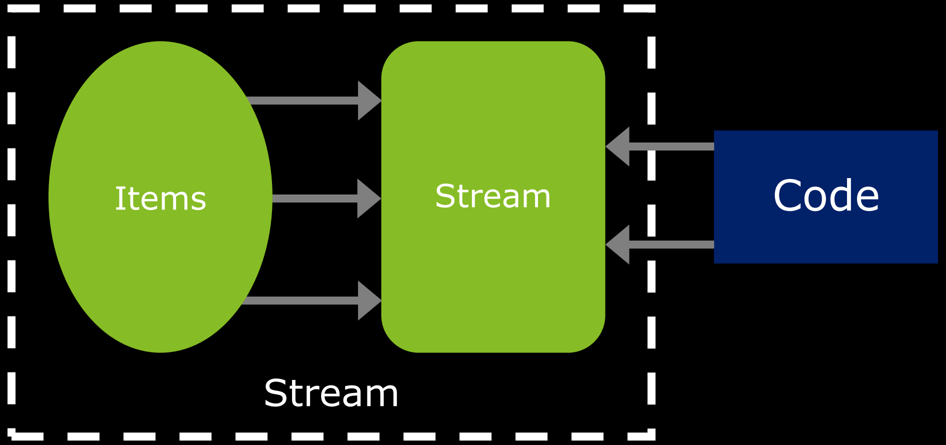
map(Student::getName)

limit(2)

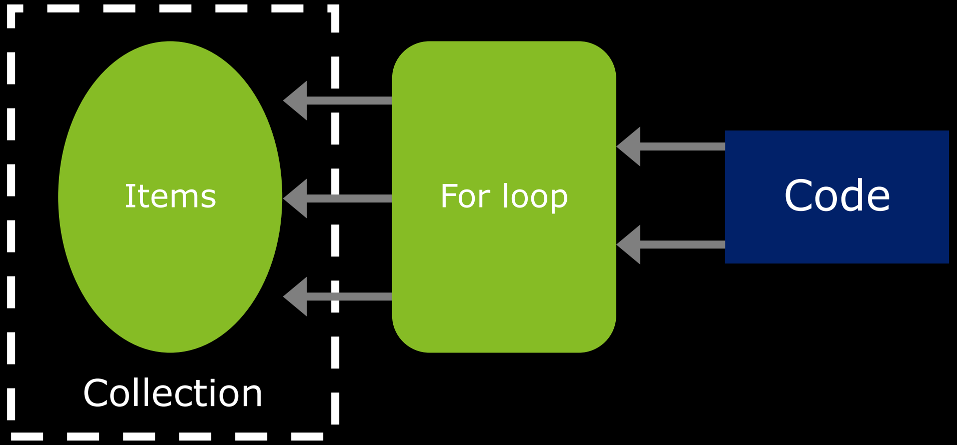
collect(toList());

Java Basics
Streams

Collections



Streams



- I will tell you what to do

- Just do it for me and optimize the work

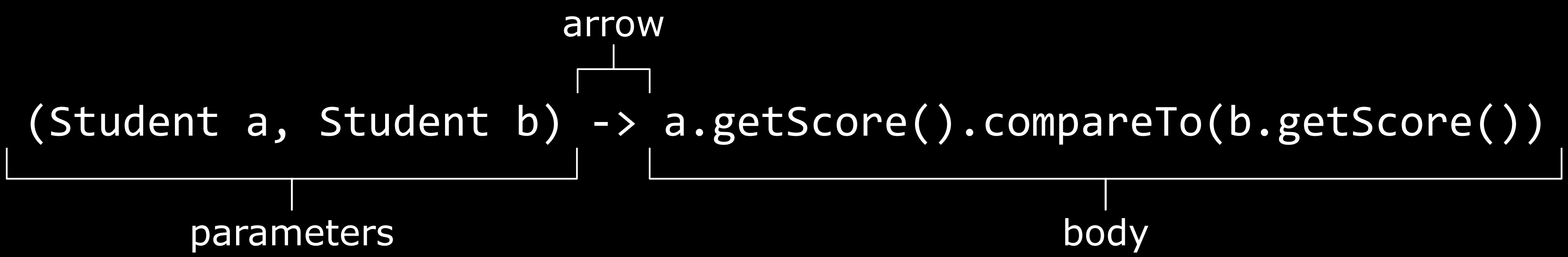
Java Basics

Exercise

- Apply an action to all elements of a list
- Create a subset list from
- Apply an action to only odd indexes of the list

Lambdas

Java Basics
Lambdas



```
(Student a, Student b) -> a.getScore().compareTo(b.getScore())
```

```
(parameters) -> expression
```

```
() -> "John"
(Integer i) -> return "John";
```

```
(parameters) -> {statements;}
```

```
() -> {}
() -> {return "John";}
(String s) -> {"John"}
```

Java Basics

Lambdas

Use Case	Example of Lambdas
A Boolean expression	(List<String> list) -> list.isEmpty()
Creating objects	() -> new Apple(10)
Consuming from an object	(Apple a) -> { System.out.println(a.getWeight()); }
Select/extract from an object	(String s) -> s.length()
Combine two values	(int a, int b) -> a * b
Runnable	newThread(() -> System.out.println(“HelloWorld”)).start();

Java Basics

Lambdas

Method references:

```
(Student a) -> a.getScore()
sort((s1, s2) -> s1.getScore().compare(s2.getScore()))
```

```
Student::getScore
sort(comparing(Student::getScore))
```

Avoid null return:

```
public String getCarModel(String licencePlate){
    Car car = carRepository.findCar(licencePlate);
    if (car != null) {
        return car.getModel();
    }
    return null;
}
```

```
public Optional<String> getCarModel(String licencePlate) {
    Optional<Car> car = carRepository.findCar(licencePlate);
    return car.map(Car::getModel);
}
```

Classes

Java Basics
Objects

An object is an instantiated entity of a class. It is identified by its:

- **State:** It is represented by attributes of an object. It also reflects the properties of an object
- **Behavior:** It is represented by methods of an object. It also reflects the response of an object with other objects
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects

Java Basics

Encapsulation

- Better control of class attributes and methods
- Class variables can be made read-only or write-only
- Increased security of data
- Modularity: the programmer can change one part of the code without affecting other parts

Java Basics
Constructors

- A constructor in Java is a special method that is used to initialize objects

- The constructor is called when an object of a class is created

- It can be used to set initial values for object attributes

Java Basics

Enums

- It is a special class that represents a group of constants

- It can, just like a class, have attributes and methods

- Cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces)

Java Basics

Generics

- Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively

- Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time

- Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements

Access modifiers

Java Basics
Modifiers

- Access modifiers
- Non-access modifiers

```
[modifier] class ABCD { …

[modifier] void method(int number) { …

[modifier] int number;
```

© 2019. For more information, contact Deloitte Portugal

Java Basics

Access modifiers

	`<default>`	`private`	`protected`	`public`
Same class	✓	✓	✓	✓
Same package subclass	✓		✓	✓
Same package non-subclass	✓		✓	✓
Different package subclass			✓	✓
Different package non-subclass				✓

Java Basics

Non-Access modifiers

`static`	Attributes and methods belongs to the class, rather than an object
`final`	Attributes and methods cannot be overridden/modified; Classes cannot inherited from it
`abstract`	Classes cannot be used to create objects; Methods do not have a body. The body is provided by the subclass
`transient`	Attributes and methods are skipped when serializing the object containing them
`synchronized`	Methods can only be accessed by one thread at a time
`volatile`	The value of an attribute is not cached thread-locally, and is always read from the "main memory"

Inheritance

Java Basics

Inheritance

- It is possible to inherit attributes and methods from one class to another.

- To inherit from a class, use the **extends** keyword.

- We group the "inheritance concept" into two categories:
  - Subclass (child) - the class that inherits from another class
  - Superclass (parent) - the class being inherited from

Java Basics
Abstract class

- Is the process of hiding certain details and showing only essential information to the user
- Abstraction can be achieved with either abstract classes or interfaces
- An abstract class can have both abstract and regular methods
- Use to achieve security - hide certain details and only show the important details of an object

Java Basics
Interfaces

- An interface is a completely "abstract class" that is used to group related methods with empty bodies

- On implementation of an interface, you must override all of its methods

- Java does not support “multiple inheritance”, but However, it can be achieved with interfaces, because the class can implement multiple interfaces

Java Basics

Polymorphism

- Allows us to perform a single action in different ways. In other words, polymorphism allows you to define one “interface” and have multiple “implementations”

Overloading

When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments.

Overriding

Is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or subtype) as a method in its superclass.

Error Handling

Java Basics
Exceptions

- When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things

- When an error occurs, Java will normally stop and generate an error message – Java will throw an exception

- The programmer must handled exceptions are to ensure that:
  - The program does not end abruptly but in a controlled manner
  - It is possible to try to recover the normal flow of program execution
  - The user is warned of an exception occurring, but in a controlled manner

Java Basics

Catching exceptions

- The try statement allows you to define a block of code to be tested for errors while it is being executed.

- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

- The finally statement lets you execute code, after try...catch, regardless of the result.

```
try {
    // block of code to try
} catch(Exception e) {
    // block of code to handle errors
} finally {
    // block of code to always execute with or without an error
}
```

```
try {
    // block of code to try
} catch(Exception e1 | AnotherException e2) {
    // block of code to handle errors
}
```

Java Basics

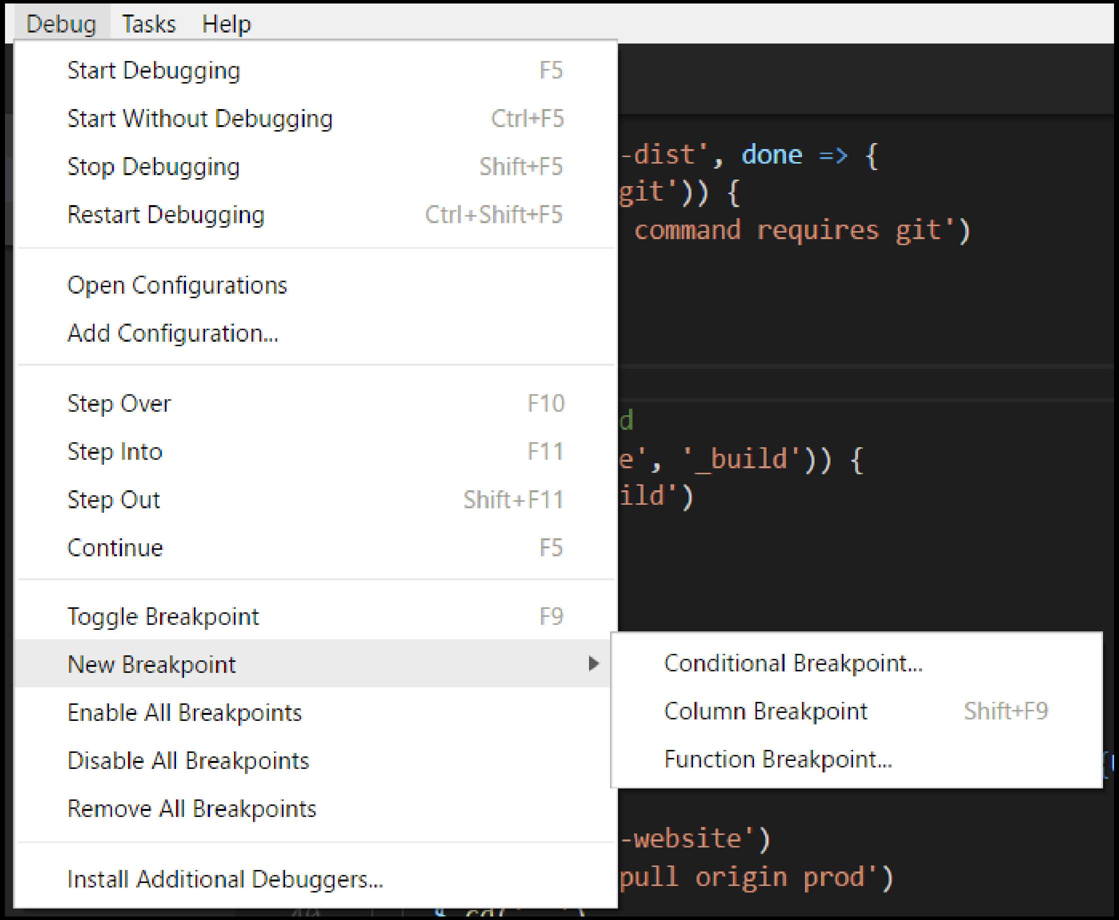
Throwing exceptions

- The throw statement allows you to create a custom error
- The throw statement is used together with an exception type
- There are many exception types available in Java: ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.
- The exception type is often used together with a custom method

Debug

Java Basics

Debug



```
-dist', done => {
git')) {
 command requires git')

d
e', '_build')) {
ild')

-website')
pull origin prod')
```