Java Workshop with Adrian

Agenda

1. What is Java?
2. Java JVM and bytecode
   i. Write, compile and run!
3. Java Java Java - syntax and symantics
   i. Data types
   ii. Class
   iii. Class: Inheritance and polymorphism
   iv. Interface
   v. Methods
   vi. Acces modifier types
   vii. Loop control
   viii. Exceptions
   ix. Collections
   x. Generics
   xi. Functions & Lambdas
   xii. Streams
   xiii. Optional
4. Core Java libraries
5. Managing multiple Java versions
6. Digging into employees-app
   - Code structure
   - Flow analysis
   - Design decisions (good and bad)
   - Good practices
   - Tests
   - Debugging

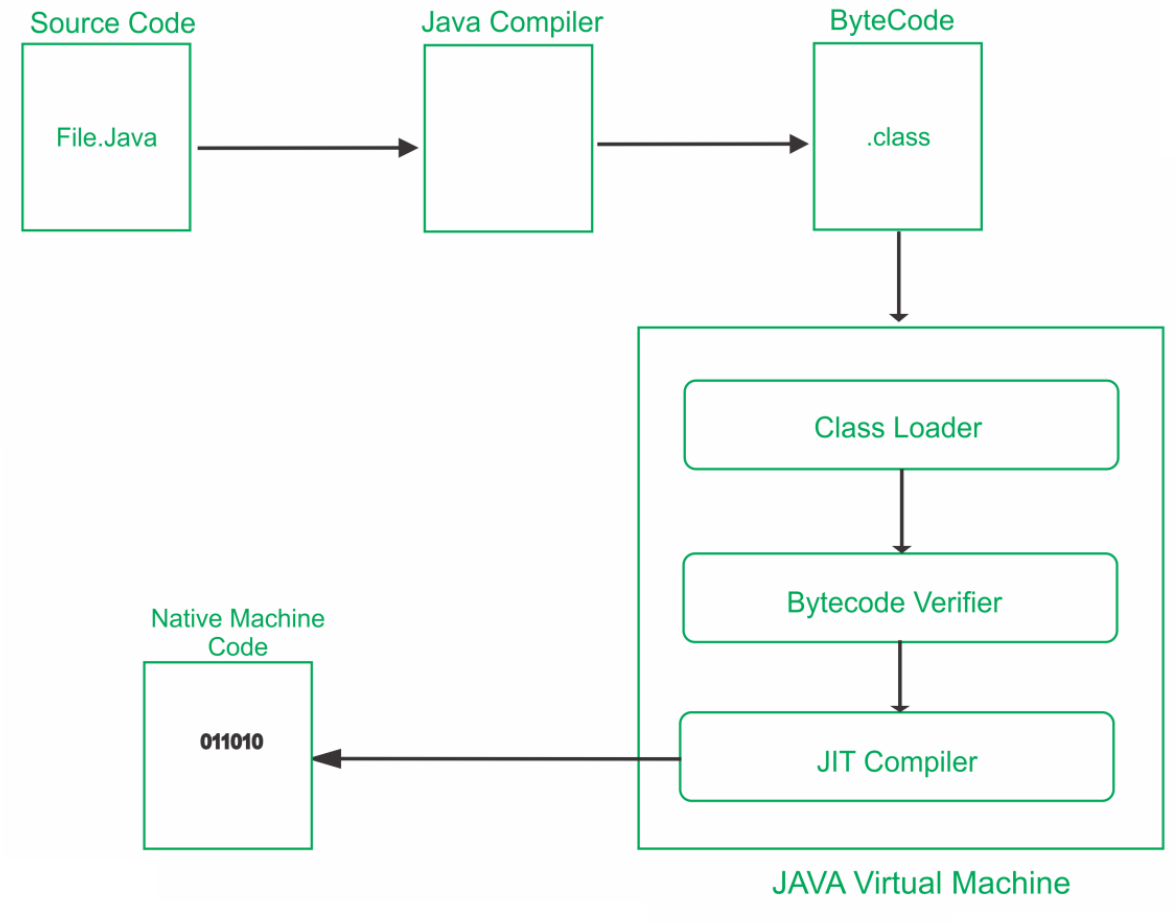
What is Java?

- Java is a **high-level**, **class-based**, **object-oriented** programming language that is designed to have as few implementation dependencies as possible.
- It is a **general-purpose** programming language intended to let programmers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need to recompile.

Source: https://en.wikipedia.org/wiki/Java_(programming_language)

Java JVM and bytecode

Java applications are typically **compiled to bytecode** that can **run on any Java virtual machine** (JVM) regardless of the underlying computer architecture.



Source: https://proandroiddev.com/optimize-java-byte-code-d37d9938729e

Write, compile and run!

Main.java

```java
public class Main {

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

$ javac Main.java
$ java Main
> Hello Java World!

Datatypes

Primitive data types (predefined by the language and named by a keyword):

- byte, short, int, long,
  - eg. long id = 5
- float, double
  - eg. double multiplier = 1,56
- boolean
  - eg. boolean result = true
- char
  - eg. char prefix = ‘a’

Reference data types (created using defined constructors of the classes), ie.:

- String message = “Hello world!”
- String error = new String(“Failure”)
- LocalDateTime now = LocalDateTime.*now*()

Class

- Class – a template/blueprint that describes the behaviour and state that the objects support.
- Object – an instance of a class, can have state and behaviour.

```
class Cat {
    private String breed;
    private String name;
    private int age;

    public Cat(String breed, String name, int age) {
        this.breed = breed;
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

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

    void bark() {
    }

    void sleep() {
    }
}


Cat cat = new Cat(“Ragdoll”, “Kitty”, 5);
cat.bark();
```

Inheritance and polymorphism

```java
abstract class Animal {
    protected String breed;
    protected String name;
    protected int age;

    protected Animal(String breed, String name, int age) {
        this.breed = breed;
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

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

    abstract boolean isHungry();

    abstract void sleep();
}
```

```java
Animal cat = new Cat("Ragdoll", "Kitty", 3);
cat.isHungry();
```

```java
class Cat extends Animal {

    private int satietyLevel;

    public Cat(String breed, String name, int age) {
        super(breed, name, age);
    }

    public Cat(String breed, String name, int age, int satietyLevel) {
        super(breed, name, age);
        this.satietyLevel = satietyLevel;
    }

    @Override
    boolean isHungry() {
        return satietyLevel < 2;
    }

    @Override
    void sleep() {
        // TODO: implement cat falling asleep
    }

    void meow() {
    }
}
```

```java
Cat cat2 = new Cat("Ragdoll", "Catty", 3);
boolean isHungry = cat2.isHungry();
cat2.meow();
```

Interface

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

```
interface Comparable<T> {
   public int compareTo(T o);
}
```

```
abstract class Animal implements Comparable<Animal> {

   // rest of implementation

   @Override
   public int compareTo(Animal o) {
      return this.age - o.age;
   }
}
```

Method

A method is a block of code which only runs when it is called.

```
class AnimalFactory {

    public static String[] supportedSpecies() {
        return new String[]{"Dog", "Cat", "Cow"};
    }

    public Animal create(String species, String name, String breed, int age){
        return null;
    }

    public Animal create(String species, String name, String breed) {
        return null;
    }

}
```

```
String[] supportedSpecies = AnimalFactory.supportedSpecies();

AnimalFactory factory = new AnimalFactory();
Animal newCat = factory.create("Cat", "Kitty", "Ragdoll");
Animal newCat2 = factory.create("Cat", "Kitty", "Ragdoll", 5);
```

Access modifiers

Access modifiers are used to set the accessibility (visibility) of classes, interfaces, variables, methods, constructors, data members, and the setter methods.

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the declared class
default	The code is only accessible in the same package. This is used when you don't specify a modifier.
protected	The code is accessible in the same package and **subclasses**.

Source: https://www.w3schools.com/java/java_modifiers.asp

Loop control

While loop

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

For loop

```
for (int i = 0; i < 5; i++) {
   System.out.println(i);
}
```

Do While loop

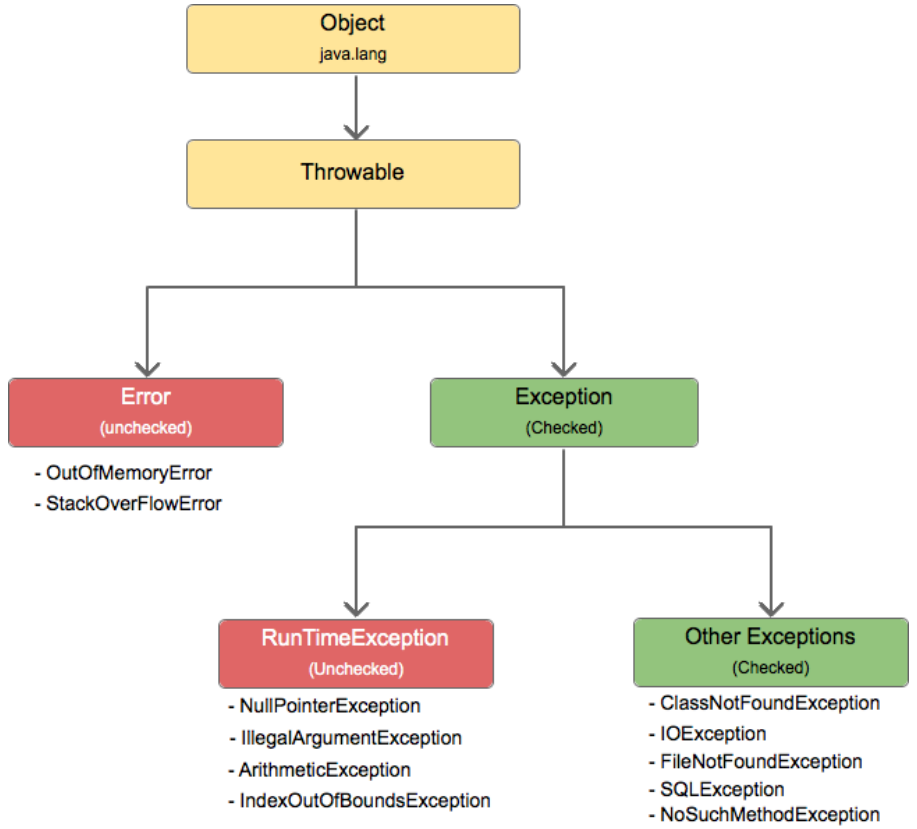
```
do {
   // code block to be executed
} while (condition);
```

For-each loop

```
String[] colors = {”Black", ”Blue", ”Green", ”Red"};
for (String i : colors) {
   System.out.println(i);
}
```

Java Exceptions

Exception - an event that occurs during the execution of a program that disrupts the normal flow of instructions.



Source: https://hashcodec.com/java-programming/exception-handling

Try-catch

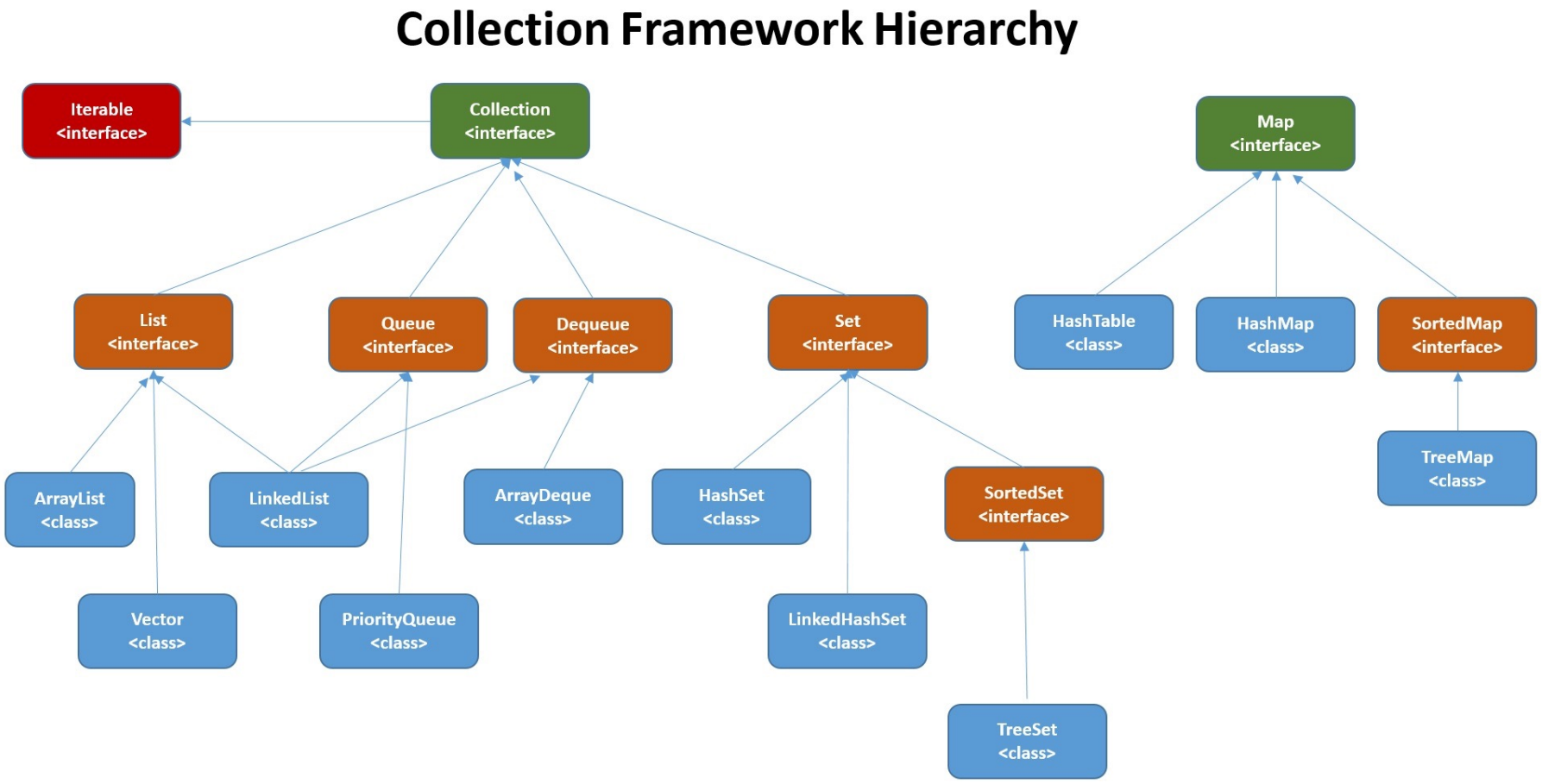
```
try {
    // Block of code to try
} catch(Exception e) {
    // Block of code to handle errors
}
```

Try-catch-finally

```
try {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]);
} catch (Exception e) {
    System.out.println(”Exception thrown.
} finally {
    System.out.println("The 'try catch' is
finished.");
}
```

```
public class Main {
    public static void main(String[ ] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Something went wrong."); }
        }
    }
}
```

Java Collections

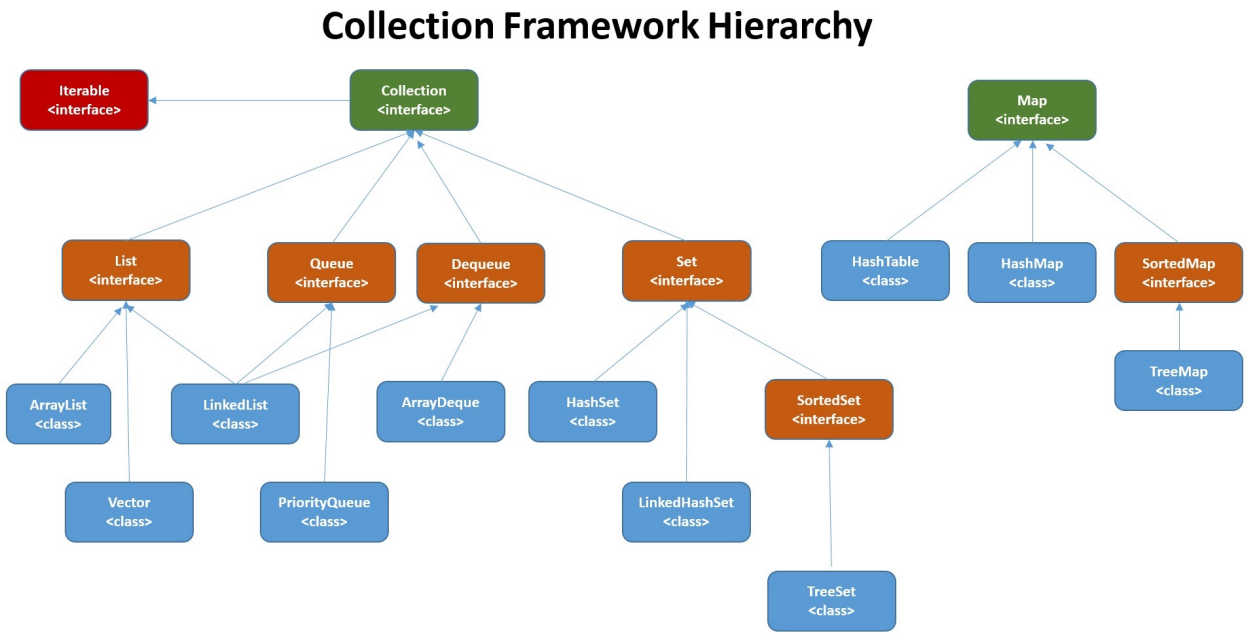


Source: https://facingissuesonit.com/2019/10/15/java-collection-framework-hierarchy/

Java Collections

```
Map<Integer, String> dayOfWeekNames = new HashMap<>();
dayOfWeekNames.put(0, "Monday");
dayOfWeekNames.put(1, "Tuesday");
var monday = dayOfWeekNames.get(0);

List<String> names = new ArrayList<>();
names.add("John");
names.add("Tony");
var john = names.get(0);
```



Java Generics

- Problem

```
List list = new LinkedList();
list.add(new Integer(1));
Integer i = list.iterator().next();
Integer i = (Integer) list.iterator().next();
```

- Solution

```
List<Integer> list = new LinkedList<>();
list.add(new Integer(1));
Integer i = list.iterator().next();
```

```
public <T> List<T> fromArrayToList(T[] a) {
    return Arrays.stream(a)
            .collect(Collectors.toList());
}
```

Java Functions & Lambdas

Any interface with a SAM (Single Abstract Method) is a functional interface, and its implementation may be treated as lambda expressions.

A lambda is an anonymous function.

```
@FunctionalInterface
public interface Function<T, R> {
    R apply(T t);
}
```

```
Function<String, String> uppercase = s -> s.toUpperCase();
uppercase.apply("string");
```

```
@FunctionalInterface
public interface Supplier<T> {
    T get();
}
```

```
Supplier<String> stringSupplier = () -> "String";
String string = stringSupplier.get();
```

```
@FunctionalInterface
public interface Consumer<T> {
    void accept(T t);
}
```

```
Consumer<Integer> intPrint = i -> System.out.println(i);
Consumer<Integer> intPrint = System.out::println;
intPrint.accept(i);
```

Java Stream

Stream - a sequence of elements supporting sequential and parallel aggregate operations.

```
int sum = widgets.stream()
    .filter(w -> w.getColor() == RED)
    .mapToInt(w -> w.getWeight())
    .sum();
```

Java Optional

Optional - a container object which may or may not contain a non-null value. If a value is present, isPresent() will return true and get() will return the value.

```
Optional<String> stringOptional = Optional.of("String");
if (stringOptional.isPresent()) {
    // do something
}

var s1 = stringOptional.orElseGet(() -> "other string");
var s2 = stringOptional.orElseThrow(() -> new RuntimeException());
var s3 = stringOptional.filter(s -> s.startsWith("S"));
var s4 = stringOptional.map(s -> s.toLowerCase());
```

Core Java libraries

- java.lang → contains fundamental classes and interfaces closely tied to the language and runtime system.
- java.io, java.nio, java.net → I/O and networking API.
- java.math → provides mathematical expressions and evaluation, as well as arbitrary-precision decimal and integer number datatypes.
- java.util → built-in Collection data structures, and utility classes, for Regular expressions, Concurrency, logging and Data compression.
- java.text → deals with text, dates, numbers and messages.
- java.security, java.crypto → provide security and encryption services.
- java.sql → access to SQL databases.

Managing multiple Java versions

- https://sdkman.io/install

```
$ sdk list java
$ sdk install java
$ sdk install 18-amzn
$ sdk use java 17.0.3-tem
```

```
================================================================================
 Vendor        | Use | Version      | Dist    | Status     | Identifier
--------------------------------------------------------------------------------
 Corretto      |     | 18           | amzn    | local only | 18-amzn
               | >>> | 18.0.1       | amzn    | installed  | 18.0.1-amzn
               |     | 17.0.3.6.1   | amzn    |            | 17.0.3.6.1-amzn
               |     | 11.0.15.9.1  | amzn    |            | 11.0.15.9.1-amzn
               |     | 8.332.08.1   | amzn    |            | 8.332.08.1-amzn
 GraalVM       |     | 22.1.0.r17   | grl     |            | 22.1.0.r17-grl
               |     | 22.1.0.r11   | grl     |            | 22.1.0.r11-grl
               |     | 22.0.0.2.r17 | grl     |            | 22.0.0.2.r17-grl
               |     | 22.0.0.2.r11 | grl     |            | 22.0.0.2.r11-grl
               |     | 21.3.2.r17   | grl     |            | 21.3.2.r17-grl
               |     | 21.3.2.r11   | grl     |            | 21.3.2.r11-grl
               |     | 20.3.6.r11   | grl     |            | 20.3.6.r11-grl
```

Alternative:

- https://www.jenv.be/