# JAVA 101

(A Whirlwind Tour)

GEETHALLADI

# JAVA WHIRLWIND TOUR

# JDK

- Oracle
- Open JDK
- Amazon

# PRO(S) OVER RUBY

- stable runtime (JVM)
- statically typed
- vast libraries
- multi threading support

# TERMINOLOGIES

- java / javac / jdk
- mvn
- spring
- junit
- hibernate

# SETTING UP LOCAL ENVIRONMENT

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-11.0.10

export PATH=${PATH}:${JAVA_HOME}/bin:/usr/local/bin/mvn
```

# HELLO, WORLD

```java
package com.learning.java.samples.level0;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello, world");
    }
}
```

```
$ ls com/learning/java/samples/level0/HelloWorld.java
-rw-r--r--  1 311909993  20   168B Jan  6 21:38 com/learning/j

$ javac com/learning/java/samples/level0/HelloWorld.java

$ export CLASSPATH=.

$ java com/learning/java/samples/level0/HelloWorld.java
hello, world
```

# PRIMITIVES

- *boolean*
- *int*
- *short / long*
- *float / double*

```
public class Primitives
{
    public static void main(String[] args) {
        long creditCardNumber = 1234_5678_9012_3456L;
        long socialSecurityNumber = 999_99_9999L;
        float pi =  3.14_15F;
        long hexBytes = 0xFF_EC_DE_5E;
        long hexWords = 0xCAFE_BABE;
        long maxLong = 0x7fff_ffff_ffff_ffffL;
        byte nybbles = 0b0010_0101;
        long bytes = 0b11010010_01101001_10010100_10010010;
    }
}
```

# BASIC OBJECT ORIENTATION

- Class Creation : PhoneNumber Example

# INTERFACES

```
public interface Vehicle {
    // all are the abstract methods.
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
```

# ABSTRACT CLASS

```
// abstract class should contain atleast one abstract method
public abstract class Benchmark {

    // abstract method
    abstract void benchmark();

    public final long repeat(int count) {
        long start = System.nanoTime();
        for (int i = 0; i < count; i++)
            benchmark();
        return (System.nanoTime() - start);
    }
}
```

# EXCEPTIONS

- *try*
- *catch*
- *finally*

```
public double[] getDataSet(String setName) throws BadDataSetEx
{
    String file = setName + ".dset";
    FileInputStream in = null;

    try {
        in = new FileInputStream(file);
        return readDataSet(in);
    }
    catch (IOException e) {
        throw new BadDataSetException();
    }
    finally {
        try {
```

# JAVA COLLECTIONS

## LIST

- *ArrayList*
- *LinkedList*

## MAP

- *HashMap*

## SET

- *HashSet*

# JAVA CONCURRENT COLLECTIONS

- *ConcurrentHashMap*
- *CopyOnWriteArrayList* (and *CopyOnWriteArraySet*)
- *BlockingQueue*
- *ConcurrentSkipListMap*

# BEST PRACTICES

- Members are always private
- Use *.equals* over ==
- Program to the interface not to the implementation
- Prefer Composition over Inheritance
- Prefer *double* over *float*
- Overriding *hashcode* and *equals*

# BEST PRACTICES

- Immutable objects
- Prefer Empty Collections instead of *null*
- Strings are value objects
- Writing doc comments for your public methods
- Program Defensively
- Avoid excessive usage of *null*
- Make sure to spend time understanding Java regular expressions

# BEST PRACTICES

- Avoid empty *catch* blocks
- Make sure the exception traces are logged
- Use Java Collections

# SOLID CLASS DESIGN PRINCIPLES

- Single responsibility principle
- Open/closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle

# USAGE OF DESIGN PATTERNS

- Builder
- Singleton
- Proxy
- Visitor
- Factory Method
- Abstract Factory

# COMPARISONS

| Ruby | Java |
|---|---|
| require | import |
| nil | null |
| module | package |
| mixin | interfaces |
| freeze | final |

# COMPARISONS

| Ruby | Java |
| --- | --- |
| attr accessors | get / Set |
| public / private / protected | public / private |
| rake | mvn |
| Array | ArrayList |
| Map / dictionary | HashMap |

# REFERENCES

- Java API documentation
- Effective Java
- Java Concurrency in Practice
- "Gang of Four" Design Patterns
- SOLID Principles