skills matter

Java Jump-start for Scripters

Alexander Worton

Civic Reactor @ Skills Matter

If you put a million monkeys on a million keyboards…

Eventually, one of them will write a **Java program**.

The rest will write **Perl programs**.

git **https://tinyurl.com/y7dkhdoy**

Obligatory biography

- I have worked in primary / secondary education since 2003 as a Qualified Teacher, Manager and Software developer.
- I am transitioning full time into software development, and am currently an MSc Computer Science student at Birkbeck, University of London where my most recent experience is in Java and Scala.

https://www.linkedin.com/in/alexander-worton/

https://skillsmatter.com/groups/10587-civic-reactor

https://tinyurl.com/y7dkhdoy

Pre-req: You should have the latest Java installed and included on your classpath

What this session will cover

Aim: Provide a small taste of Java programming and highlight Civic Reactor's mission

1. *Shameless Plug* - How Civic Reactor uses Java as part of a broad range of technologies to develop applications for community enrichment and skills development
2. A fast paced overview of Java and where it makes sense
3. An overview of Java datatypes
4. A jumpstart in writing, naming, compiling and running Java applications

- OOP and Java's architecture: Inheritance, Encapsulation, Polymorphism and message passing

Civic Reactor

We provide an environment to enable developers and other IT professionals to work collaboratively on open source projects which provide civic enrichment.

- To develop the skills of those working on projects
- To allow people to work on technologies that they don't get to in their jobs
- To provide exposure to collaborative work

Indicative Civic Reactor Projects

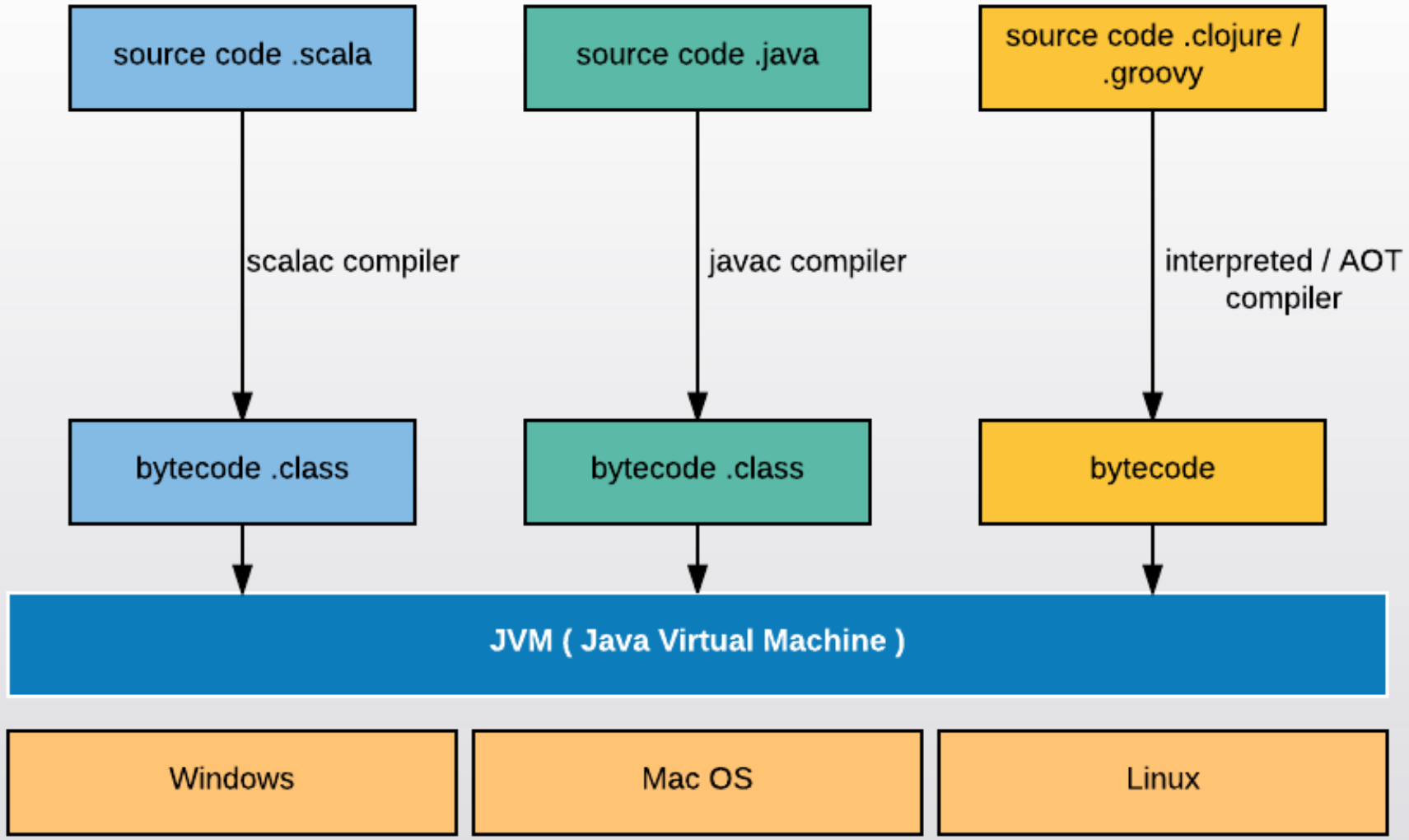
- Meals on wheels
- Care home app
- Civic Reactor website / developer-project matching app
- NHS-CancerWaitingList
- MasterBus

Technologies: Java, Spring, Angular, Typescript, Javascript, React, Ionic, Firebase, MySQL, Neo4j, Python, Akka, Scala, Kafka …

What does Java give us?

- Java is an enterprise language designed for portability: write once, run anywhere. It features:
  - Strong Typing
  - Good concurrency support for multithreading
  - 'Classic' OOP and partial Functional paradigms (v8+)
  - Automatic garbage collection
  - An extensive community library
  - Stabilisers to support programmers in building scalable applications

2/4 A fast paced overview of Java and where it makes sense



Java and where it makes sense

So where does a robust, enterprise grade language make sense?

Backend server code

Mainframes

Cross platform GUI Apps (JavaFX)

Embedded devices

Android Mobile (Kotlin?)

Java criticisms

- Backwards compatibility results in poorer feature implementation (Generics, Functional Interfaces…)
- Garbage collector runs when it feels like it. This can result in collection at inopportune moments…
- Simple and complex types being treated differently
- built in Java collections (Guava)
- …and a whole lot more, including …

Java criticisms

- It's verbose. I mean, really verbose.

```scala
/**
 * @author Alexander Worton.
 */
case class Poso(name: String, age: Int, hip: Boolean)
```

```java
/**
 * @author Alexander Worton.
 */
public class Pojo {

    public String name;
    public int age;
    public Boolean hip;

    public Pojo(String name, int age, Boolean hip) {
        this.name = name;
        this.age = age;
        this.hip = hip;
    }
}
```

```java
/**
 * @author Alexander Worton.
 */
public class Pojo {

    private String name;
    private int age;
    private Boolean hip;

    public Pojo(String name, int age, Boolean hip) {
        setName(name);
        setAge(age);
        setHip(hip);
    }

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Boolean getHip() {
        return hip;
    }

    public void setHip(Boolean hip) {
        this.hip = hip;
    }
}
```

Why Java?

	Skill / Job Role(Links to historical trends & salary statistics)	Rank3 Months to 23 Jun 2017	Rank ChangeYear-on-Year	Median Salary3 Months to 23 Jun 2017	Median Salary % ChangeYear-on-Year	Matching Permanent Job Ads3 Months to 23 Jun 2017	Live Job Vacancies
1	SQL	4	0	£45,000	-	18910 (21.22%)	2558
2	JavaScript	5	0	£45,000	-	16453 (18.46%)	2541
3	C#	6	+2	£45,000	-	15084 (16.93%)	2169
4	Java	9	+4	£55,000	-	14130 (15.86%)	2055
5	Python	23	+7	£55,000	-	6970 (7.82%)	1097
6	PHP	55	-9	£40,000	-	4622 (5.19%)	829
7	C++	65	-4	£45,000	-5.26%	4165 (4.67%)	671
8	T-SQL	75	+3	£45,000	-	3281 (3.68%)	362
9	C	90	+4	£47,500	-	2919 (3.28%)	490
10	Ruby	99	-8	£55,000	-	2726 (3.06%)	412
11	PowerShell	124	+27	£47,500	-	2339 (2.62%)	294
12	Bash Shell	184	+17	£55,000	+4.76%	1660 (1.86%)	247
13	Perl	190	-33	£55,000	+4.76%	1638 (1.84%)	231
14	Scala	206	+27	£67,500	+3.84%	1477 (1.66%)	253
15	PL/SQL	227	-31	£50,000	-	1356 (1.52%)	120

https://www.itjobswatch.co.uk/default.aspx?page=1&sortby=5&orderby=0&q=&id=900&lid=2618 Accessed 23 June 2017

Java quiz

- How is the object lifecycle managed in Java?
  - *The garbage collector runs collecting and freeing up memory from orphaned objects which have no references to them*
- What are the two categories of types in Java called?
  - *Simple and Complex types*
- What is Java source code compiled into?
  - *Bytecode which is JIT compiled into platform specific code at runtime*

Build your Hello World app

Don't forget to modify the program to output "Hello World!"

```
public class Hello {

	public static void main(String[] args) {

		System.out.println("message");

	}

}
```

Tip: the file name must be the same as the class name

Tip: once saved, open a terminal, navigate to the file location and compile: **`javac Hello.java`**

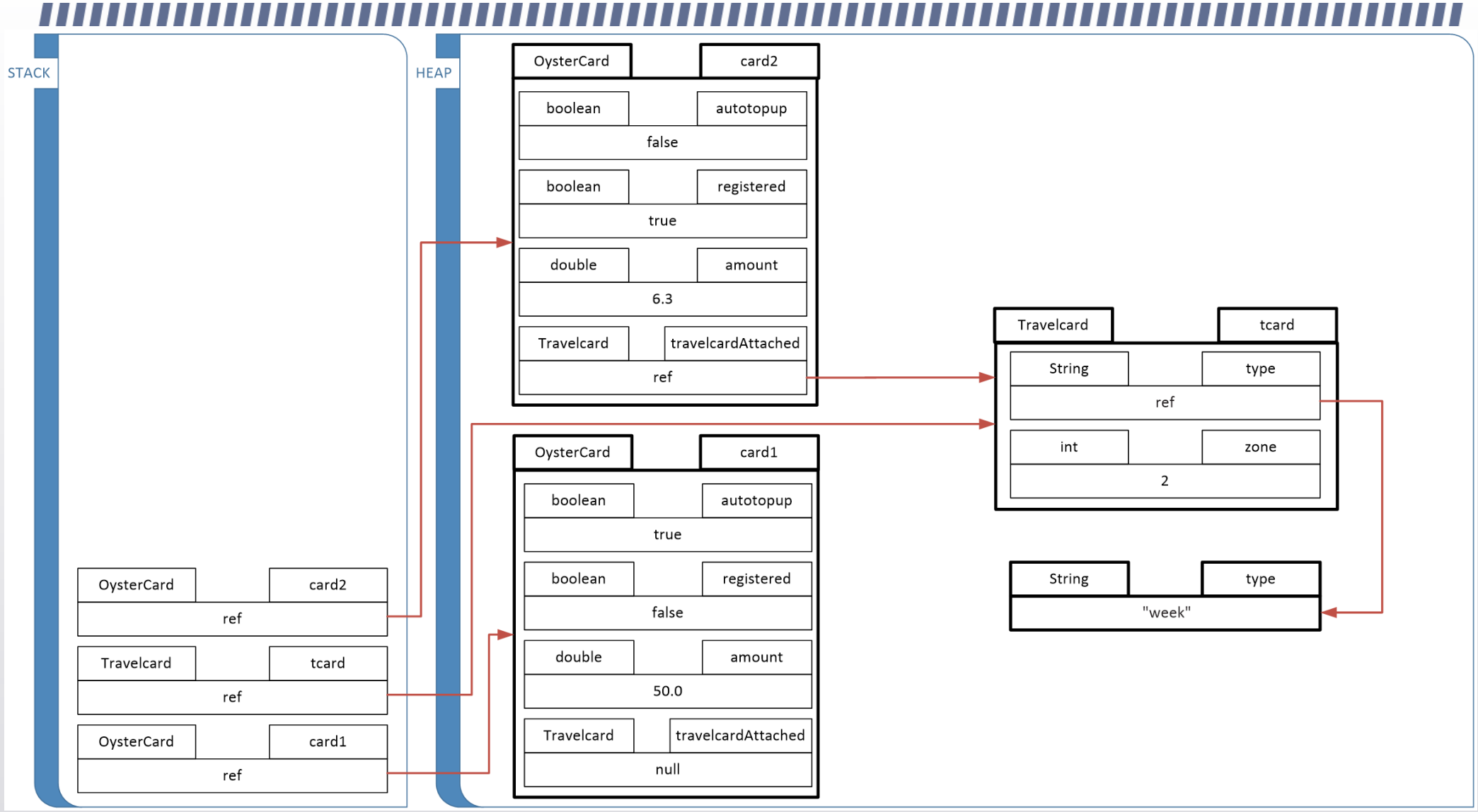
Tip: to run the compiled bytecode, navigate in terminal to the same folder as the file and execute **`java Hello`**

https://tinyurl.com/y7dkhdoy src/01FirstJavaProgram

Datatypes

- Java has two sets of datatypes; simple types and complex types
- Simple types hold the value of the type in the direct memory space allocated for the value.
- Complex types usually exceed this available space, so instead store their values on the heap and use the direct value space to hold a reference to the memory location where the value is stored.
- This is best explained through a diagram.

3/4 An overview of Java datatypes



Datatypes

- boolean
- byte
- short
- char
- int
- long
- float
- double

- Boolean
- Byte
- Short
- Character
- Integer
- Long
- Float
- Double
- String

Equivalence a == b

- In Java, the == comparison operator makes a comparison on the value. This is fine for simple types, but in the case of complex types it will only return true where this reference value matches; the values are a reference to the same object on the heap.
- Complex types use an equals method for comparison instead. This is inherited from the base object and can (and should) be overridden to check equivalence based on the relevant class fields.

ob1.equals(ob2)

Method calls obj.doThing("Some", "Stuff");

- When a method call is invoked, arguments may be supplied to the defined parameters in the method signature.
- These are passed by value; copied for local use inside the scope of the method.
- Simple types therefore are a copy of the value and do not have any effect outside the scope of the method.
- Complex types however are a reference to an object and changes to them affect the object outside of that scope.

Java Datatypes quiz

- What are the two categories of types in Java called?
  - *Simple and Complex types*
- How are each of the types named?
  - *Simple types are lower cased, Complex types are Pascal cased*
- How many Java datatypes can you remember?
  - *boolean byte short char int long float double*
  - *String*

Your Datatypes Turn

- Create an **App.java** file which contains a class (named App of course) with an entry point to the program. ***public static void main(String[] args){}***
- Declare and initialise a number of variables, both simple and complex.
- int age = 5; boolean isHappy = true; Integer legs = new Integer(2);
- Try out equivalence checks. *if (a == b) System.out.println("a == b");*
- Make sure your program compiles. Bonus: Print the data to the console.
- Hint: System.out.println("Age: " + age);

https://tinyurl.com/y7dkhdoy src/02Datatypes

Classes

- A class is a template which combines *data* and *functionality* from which objects may be *instantiated* using the 'new' keyword.
- This allows code *reuse* and adherence to the **DRY** principle.
- Classes exhibit *encapsulation*. This means that implementation details are encapsulated inside the class and *hidden* from other parts of the system. This reduces *complexity* and improves *robustness*.
- A class can only be interacted with via its public API - the methods and fields which are published via access modifiers. This is achieved through message passing, such as a message to call a method or set a field value.
- Public methods and fields encapsulated in a class can be accessed through the dot notation ObjectName.method(parameters…); or ObjectName.field;

Rules for writing classes

- In Java, the filename must match the name of the class.
  - class Foo {} must be in a file named Foo.java
- A method is a function which resides within a class. It should be written in camelCase.
- A Java application must have an entry point. That is a public, static method named main which accepts an array of command line arguments.
- A class name should be written in PascalCase.

Packages

- A package is like an Angular 4 module. It collects and groups a set of related classes. It can also determine an API through which the package can be accessed in the same way that a class can encapsulate data and functionality.
- If no package is defined, the class will be placed into the default package.
- This makes it harder to unit test, so avoid using the default package.

Packages

- Copy the Hello.java file to HelloPackage.java and change the class name to match
- Add at the top of the file the line package com.civicreactor;
- place the file in the directory structure com\civicreactor\HelloPackage.java
- compile the file using javac
- run the file from the root folder: java com.civicreactor.HelloPackage

```
λ tree
Folder PATH listing
Volume serial number is A233-EDA2
C:.
└───com
    └───civicreactor
```

git **https://tinyurl.com/y7dkhdoy** src/03Packages

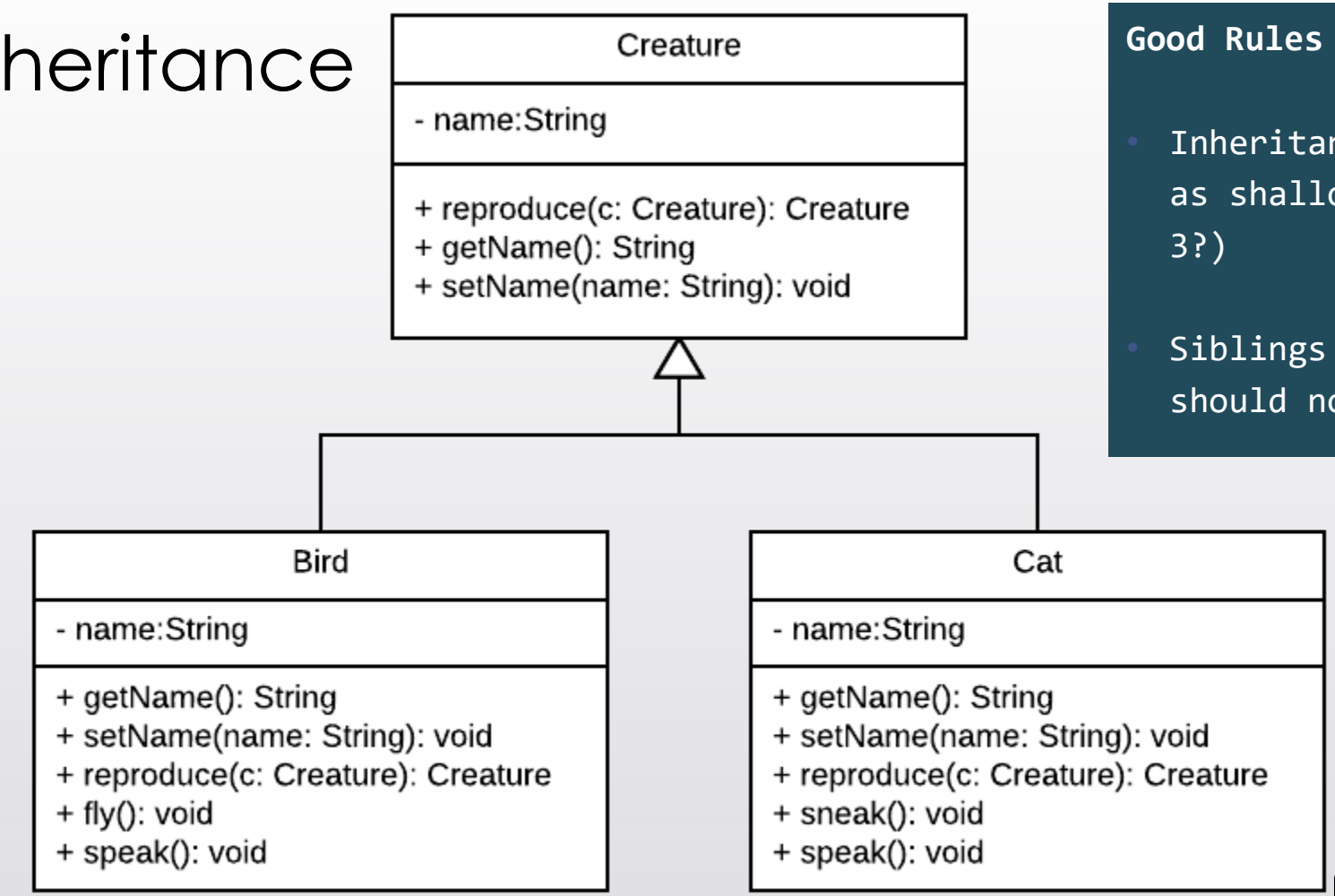
Inheritance

- Inheritance allows for a form of *Polymorphism* (*subtyping*), where the same operations can be performed on different types.
- It is important to consider the '**L**' in **SOLID** principles when utilising inheritance in order to build scalable, maintainable applications.
- A general rule of thumb is to favour *composition* over *inheritance*. This is because inheritance can be abused and results in a much more tightly coupled, less flexible means of providing extension to behaviour.
- Many have trodden this path before us. Go study design patterns to round out your toolbox.

"The problem with object-oriented languages is they've got all this implicit environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle."

- Joe Armstrong (creator of Erlang)

Inheritance



Good Rules to live by

- Inheritance hierarchies should be as shallow as possible. (2 levels? 3?)
- Siblings in different tree paths should not share characteristics.

git **https://tinyurl.com/y7dkhdoy**

Interfaces and abstract classes

- An interface is a contract for the API through which you will interact with objects.
- An abstract class provides full or partial implementation of behaviour
- Interfaces and Abstract classes cannot be *instantiated*
- Different keywords are used. A concrete class *implements* one or more interfaces, and it *extends* a single class (abstract or concrete)

Implementing an interface

```
public class Cat implements Creature {
        private String name = "unnamed";
        public Cat(String name) {
            setName(name);
        }

        @Override
        public Creature reproduce(Creature other) {
            Cat newCat = new Cat();
            newCat.setName(combineNames(other, this));
            return newCat;
        }
        …
}
```

Instantiating and using the class

```
Creature first = new Cat("Purrdey");
Creature second = new Cat("Tiddles");

Creature spawn = first.reproduce(second);

System.out.println(spawn.getName());
```

Extending a class

```
public class Cat extends AbstractCreature {
        private String name = "unnamed";
        public Cat(String name) {
            setName(name);
        }

        @Override
        public Creature reproduce(Creature other) {
            Cat newCat = new Cat();
            newCat.setName(combineNames(other, this));
            return newCat;
        }
        …
}
```

Instantiating and using the class

```
Creature first = new Cat("Purrdey");
Creature second = new Cat("Tiddles");

Creature spawn = first.reproduce(second);

System.out.println(spawn.getName());
```

Your Inheritance turn

Write a class that implements the following interface, instantiate an object while programming to the interface and print out the name to the console using the provided method PrintPerson(Person p)'

Create another concrete implementation which returns the name in a different format e.g. instead of "FirstName SurName", you could return "Surname, FirstName", amend your main entry point to output from both classes using the same names.

This demonstrates how programming to an interface allows for easy substitution of behaviour without modifying the original class implementation.

```
private static void PrintPerson(Person p) {
    System.out.println(p.getName());
}
```

```
public interface Person {
    void setFirstName(String fName);
    void setSurName(String sName);
    String getName();
}
```

Extension: Create an abstract class in between the interface and the concrete classes to move the shared setter functionality and default getter behaviour into. Override the getter in the second concrete class.

git **https://tinyurl.com/y7dkhdoy** src/05Inheritance

Where do I go from here?

Aside from straight into another language...

https://docs.oracle.com/javase/8/docs/api/
https://www.codecademy.com/learn/learn-java
https://www.udemy.com/java-the-complete-java-developer-course/
http://pluralsight.com http://lynda.com

London Java Community (Here right now, in another room doing a more advanced talk on Java).

Indicate an interest in more talks and topics from ***Civic Reactor***.

I can speak about: OOP, SOLID principles, Design Patterns, Data Structures and Algorithms, Functional Programming, Java, Scala, Csharp, TDD / BDD, SQL databases, Information Systems, XSL, computing fundamentals, hobby electronics / arduino etc...