Java #1

Sommaire

- Pourquoi Java ?
- Présentation
- Syntaxe

Pourquoi Java ?

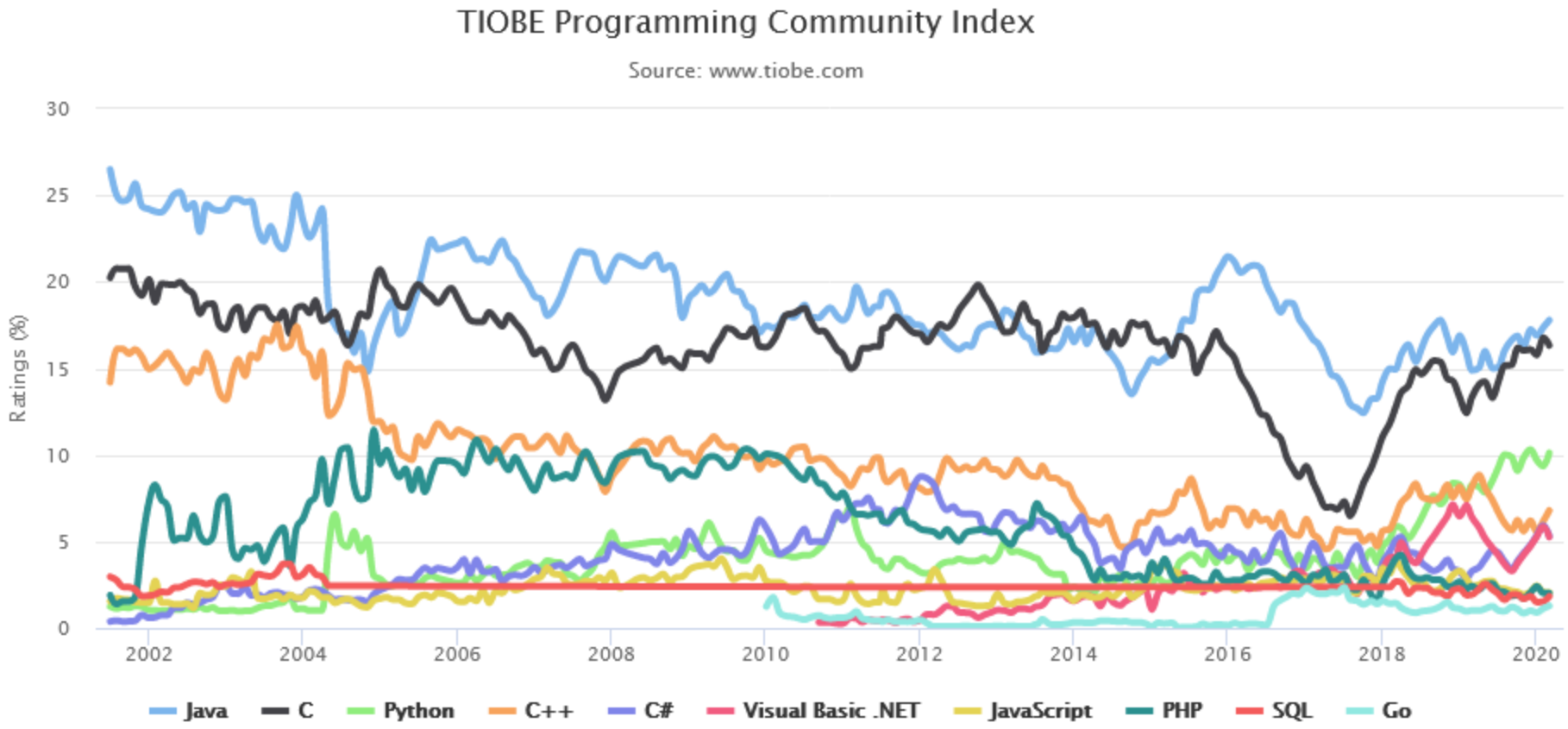
Pourquoi Java ?

- Standard industrie logicielle
- Parti pris philosophique
- Porte ouverte

Pourquoi Java ?

- Standard industrie logicielle
  - 1er TIOBE (devant C, Python, C++)
  - Lingua-franca du développement SI
  - Précurseur sur les problématiques de mise à l'échelle (machine et homme)
  - Garanties de stabilité *et* de développement
  - Très forte communauté professionnelle
  - Open Source

Pourquoi Java ?



Pourquoi Java ?

- Parti pris philosophique
  - "Reader first"
  - Travail en communauté

Pourquoi Java?

- Porte ouverte
  - Java en tant que compétence fondamentalle
  - Porte d'entrée vers l'ecosystème JVM

Présentation

(de Java)

Présentation

- 1991: Oak, crée par James Gosling - Sun Microsystems
- 1995: JAVA 1.0
- 2007: Java devient open-source
- 2010: Oracle achète Sun Microsystems

Présentation

- Langage généraliste
- Syntaxe "C-Style"
- Orienté objet
- Mémoire managée
- Code "simple"

Présentation

- Parti pris "Reader first"
  - Promotion du travail en équipe
  - Correct > Explicite > Concis
  - Abstraction des idiosyncrasies

Présentation

- Engagement de compatbilité:
  - Code once
  - Run everywhere
  - Forever...ish

Présentation

Marché:

- ~~Applet / browser~~
- Prototypage
- ~~Desktop~~
- Android
- Server Side
- Procédure Stockée / scripting Oracle
- Scripting

Présentation

Code "simple":

- Java 5: Generics
- Java 7: Generic Type Inference (Diamond Operator)
- Java 8: Lambda, Method Ref., Default Methods
- Java 9: Modules
- Java 10: Local type inference

Présentation

Performance

- CPU: même ordre de grandeur que C/C++
- Empreinte mémoire élevée
- Quelques bibliothèques / framework dispendieuses
- Compilation AoT

Présentation

Ecosystème

- Communauté vaste et qualitive
- ... pour les professionnels.
- Outillage solide

Syntaxe

- Syntaxe similaire à C/CP++
- Types primitifs: int, float, boolean...
- Types boxés: Integer, Float, Boolean...
- Exceptions.

Classes

- Java est un langage OO à classes
- Une classe contient un ensemble de
  - Propriétés,
  - Méthodes
  - Code statique

Classes

```
public class MyClass {

    private String myField = "test";

    public String getMyField() {
        return myField;
    }

    static {
        final var myClass = new MyClass();
        myClass.getMyField();
    }
}
```

Visibilités

- `public` : accessible à tous.
- ``: "default" ou "package": accessible aux sous-classes et aux classes du même package.
- `protected` : accessible aux sous-classes.
- `private` : accessibles aux instances de la classe.

Propriétés

```
@Column
private String name = "World";
```

Méthodes

```
@Transactional
protected String commit(final String value) {
    // Body: do stuff
    return value + "--";
}
```

Enum

```
public enum Color {
    RED,
    ORANGE,
    YELLOW,
    GREEN
}
```

Enum

```
public enum Color {
    YELLOW(255, 255, 0),
    ORANGE(255, 165, 0),
    GREEN(0, 255, 0),
    RED(255, 0, 0);

    public final int red;
    public final int green;
    public final int blue;

    Color(final int red, final int green, final int blue) {
        this.red = red;
        this.green = green;
        this.blue = blue;
    }
}
```

Point d'entrée

```
public static void main(final String... args) {
     Animal a = new Animal(4, "meows");
     Dragon d = new Dragon(4, "grrrrrrrrr");
}
```

Boxing

- Primitive <=> Boxed

```
Integer x = new Integer(7);
int y;
Integer z;
y = x - 1; // Auto-boxing.
z = x * y; // Auto-unboxing.
```

Class Exemple

```
// A public class
public class MyClass {

    // Some properties
    private int someInt;
    private String aString;

    // Default Constructor
    public MyClass() {
        this(42, "42");
    }

    // Overloaded constructor
    public MyClass(final int x, final String str) {
        someInt = x;
        aString = str;
    }

    //Some methods
    @Override
    public String toString() {
        return "A MyClass instance"
    }
}
```

Héritage

- Héritage de classe simple (`extends`).
- Héritage multiple d'interfaces (`implements`).
- Classes/méthodes abstraites (`abstract`).

Héritage

```
public class A {

}

public class B extends A {

}

public static void main(final String... args) {
    final var a = new A();
    final var b = new B();

    var aia = a instance a; // true
    var aib = a instance b; // false

    var bia = b instance a; // true
    var bib = b instance b; // true
}
```

Héritage

```
public abstract class A {

}

public class B extends A {

}

public static void main(final String... args) {
    final var a = new A(); // does not compile
}
```

Héritage

```
public interface A {
    void mustBeImplemented();
}

public class B implements A {
    void mustBeImplemented() {}
}
```

Demo!