Programming and Algorithms II

Lecture 2: Java Basics

Nicolas Wu

nicolas.wu@bristol.ac.uk

Department of Computer Science
University of Bristol

Hello, Java!

C and Java

- C and Java share much of their syntax

types and data	`int double char [] 42 42.0 ‘c’ “string”`
operations	`+ - * % < > <= >= == != && || ! += *= =`
control structures	`if else switch while for break continue`
procedures	`f(x) return`
delimiters	`{ } ;`
comments	`/* inline comment */ // postline comment`

wait, isn’t this just C?

Java is like C except …

- ➕ Classes and objects
- ➕ Exception handling
- ➕ Garbage collection
- ➕ Platform independent
- ➖ Pointers

These features lead to very different programming styles!

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Hello, Java

```
class HelloJava {

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

}
```

Tools

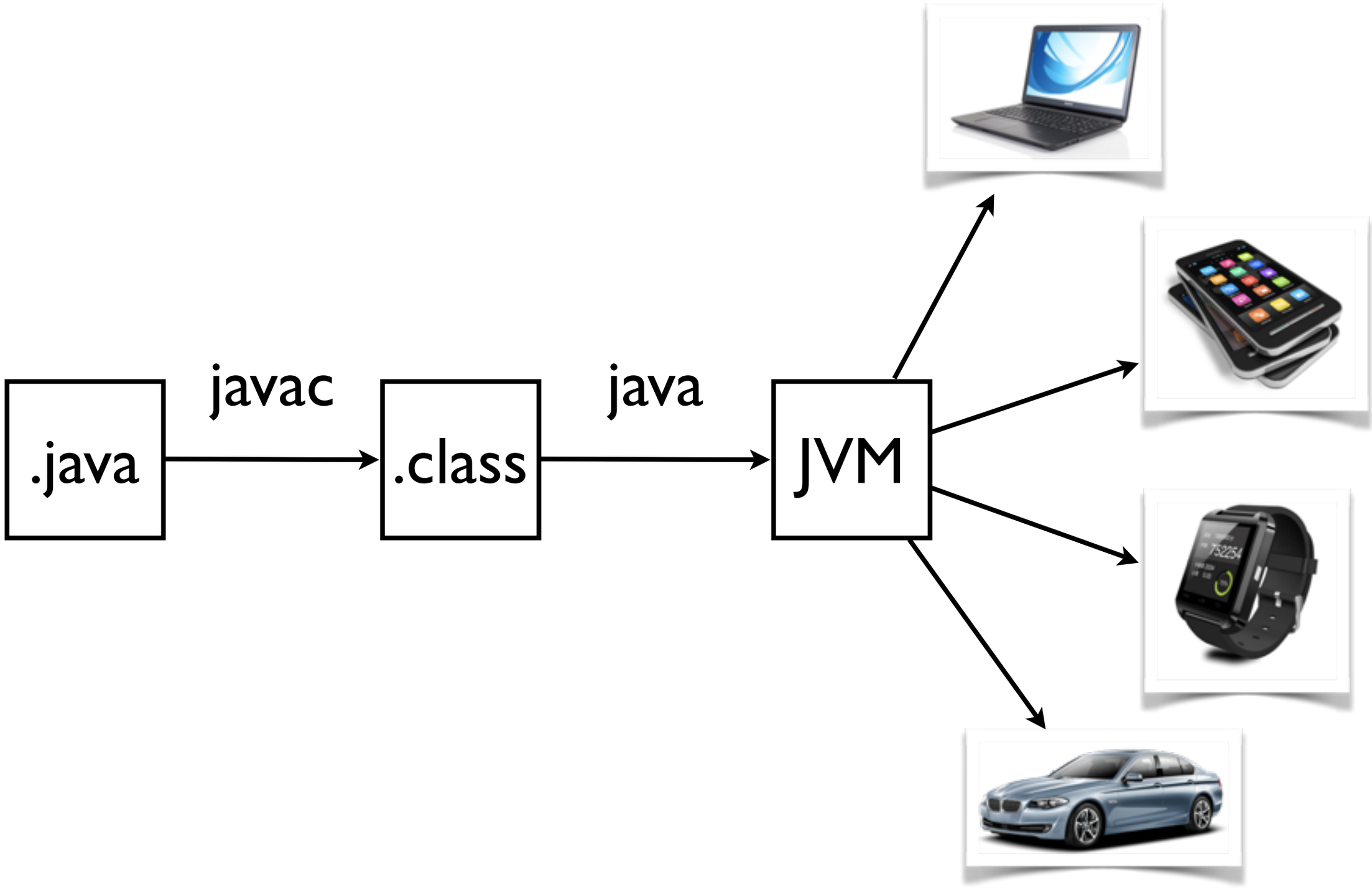
Hello, Java!

- Create a new HelloWorld.java file
- Implement the standard “Hello, Java!”
- Compile to HelloWorld.class with javac
- Run HelloWorld with java

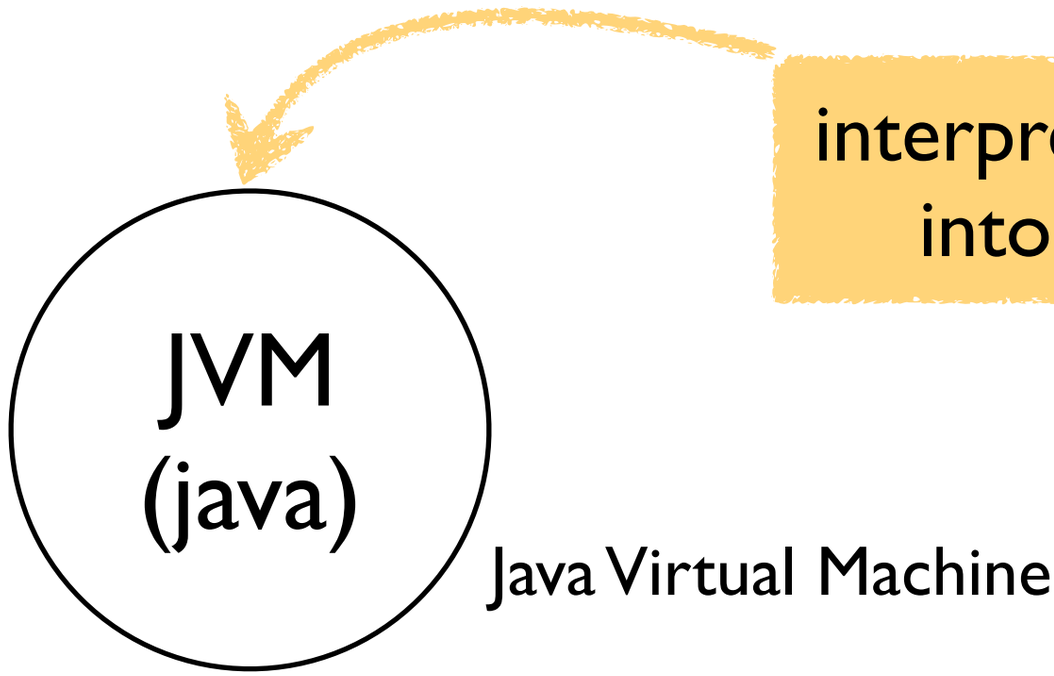
javac vs java

compiler	interpreter
static	dynamic
compile errors	run-time exceptions
.java	.class

Toolchain



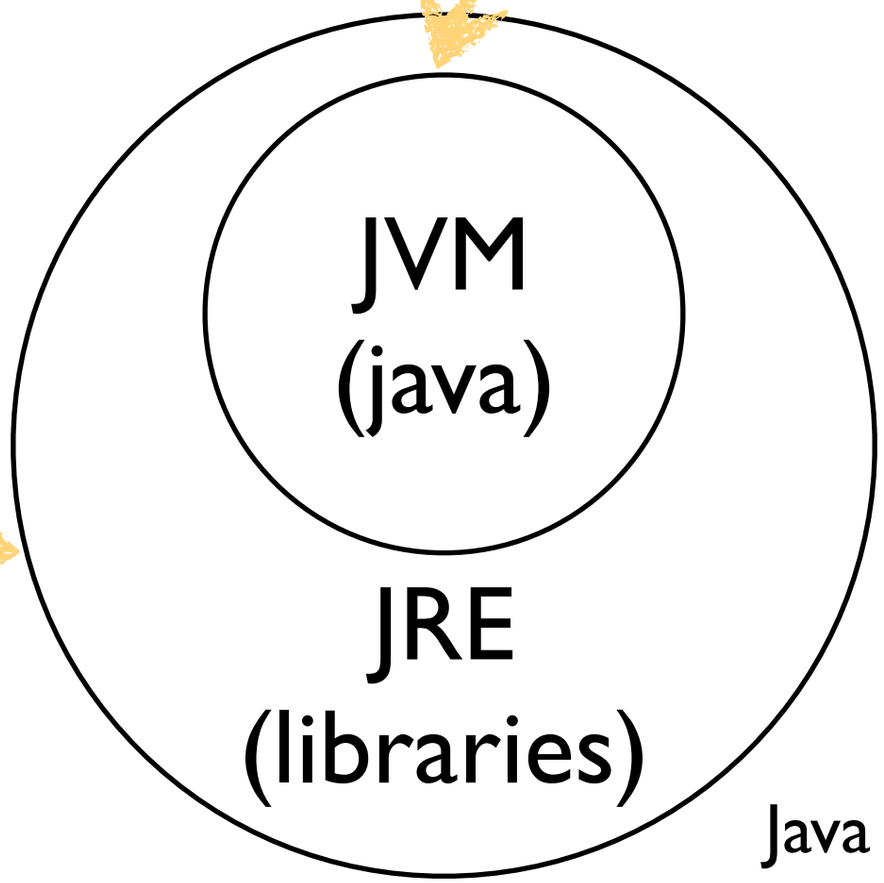
JVM, JRE, JDK, and JFK



JVM, JRE, JDK, and JFK

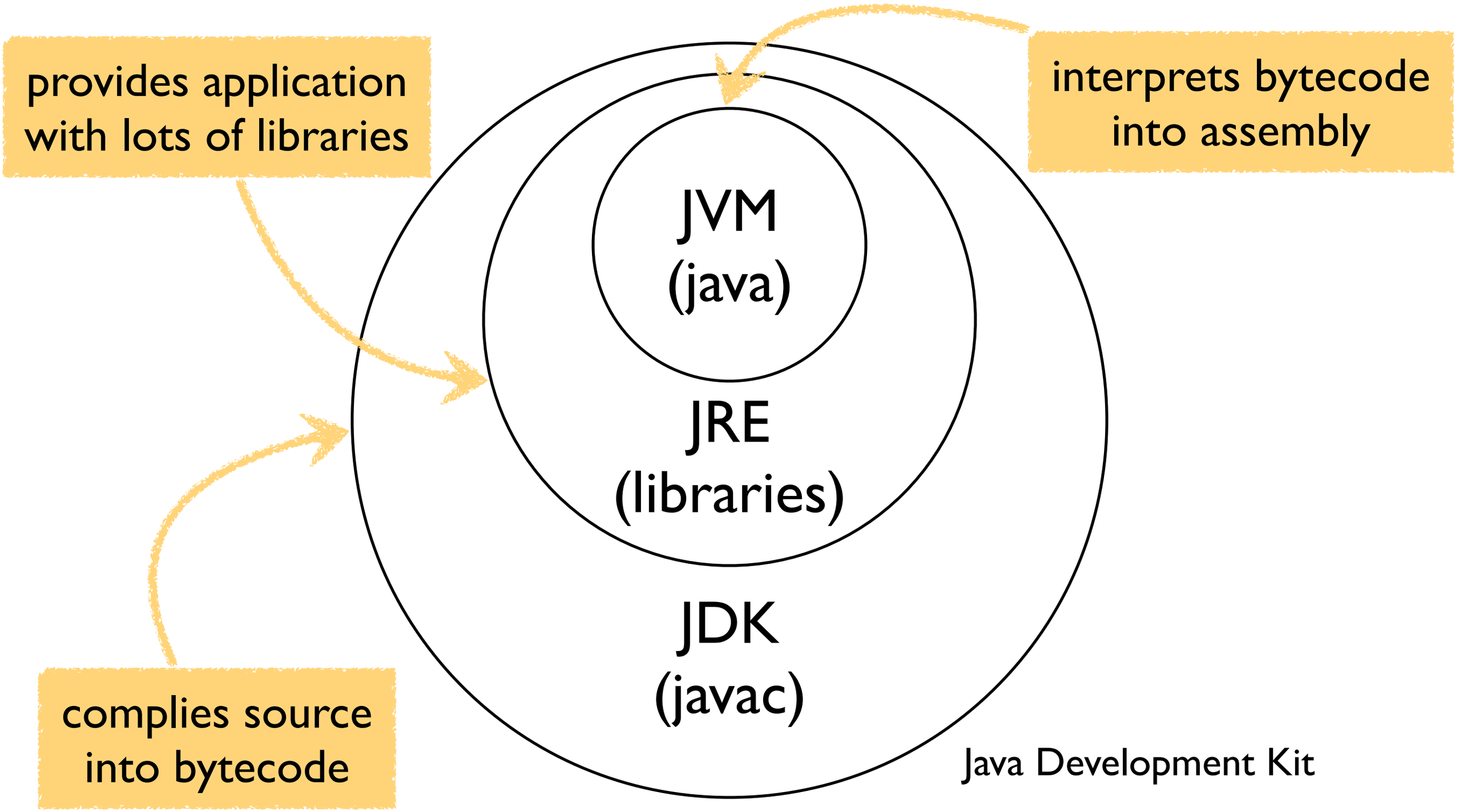
provides application with lots of libraries

interprets bytecode into assembly



Java Runtime environment

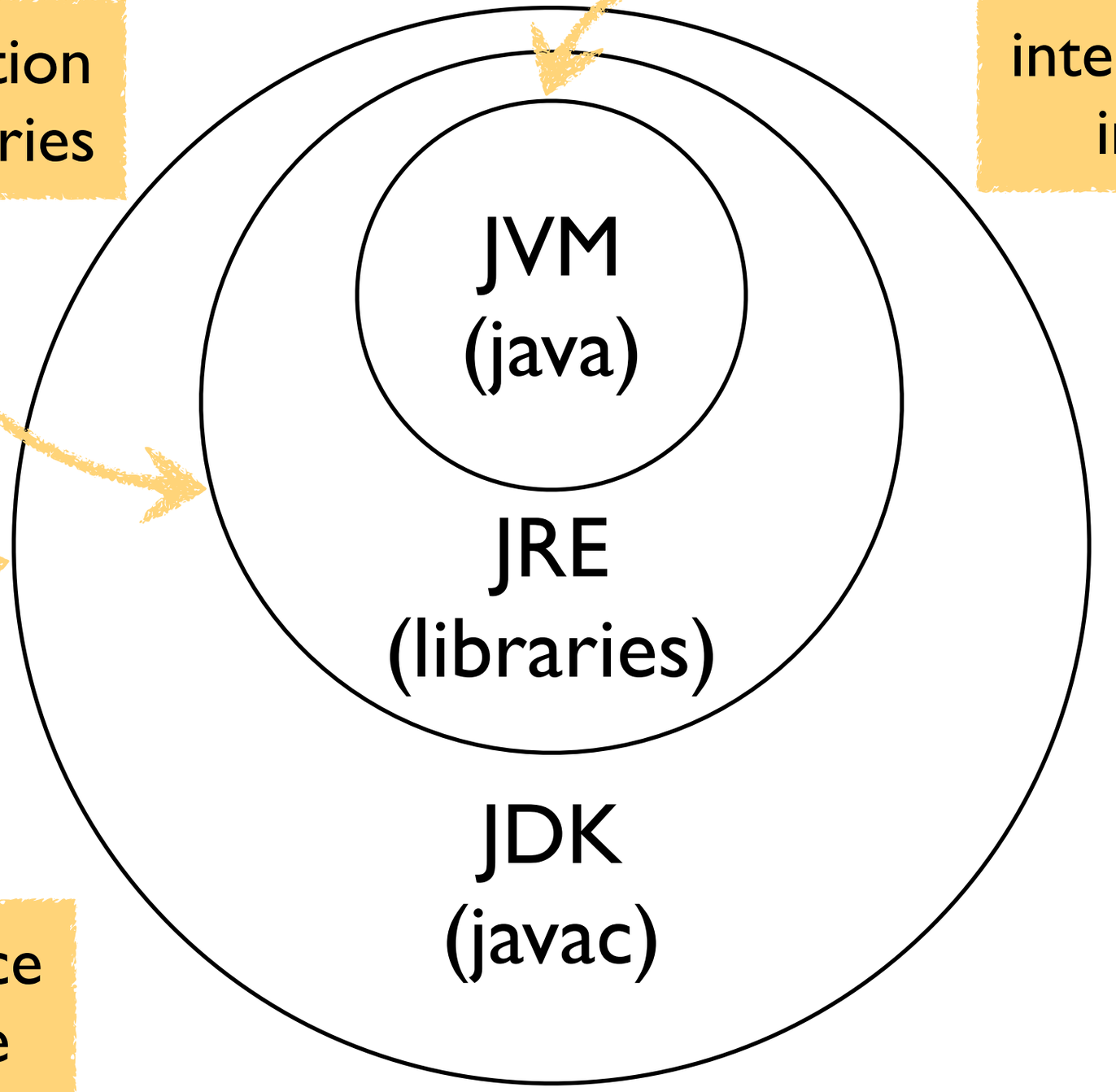
JVM, JRE, JDK, and JFK



JVM, JRE, JDK, and JFK

provides application with lots of libraries

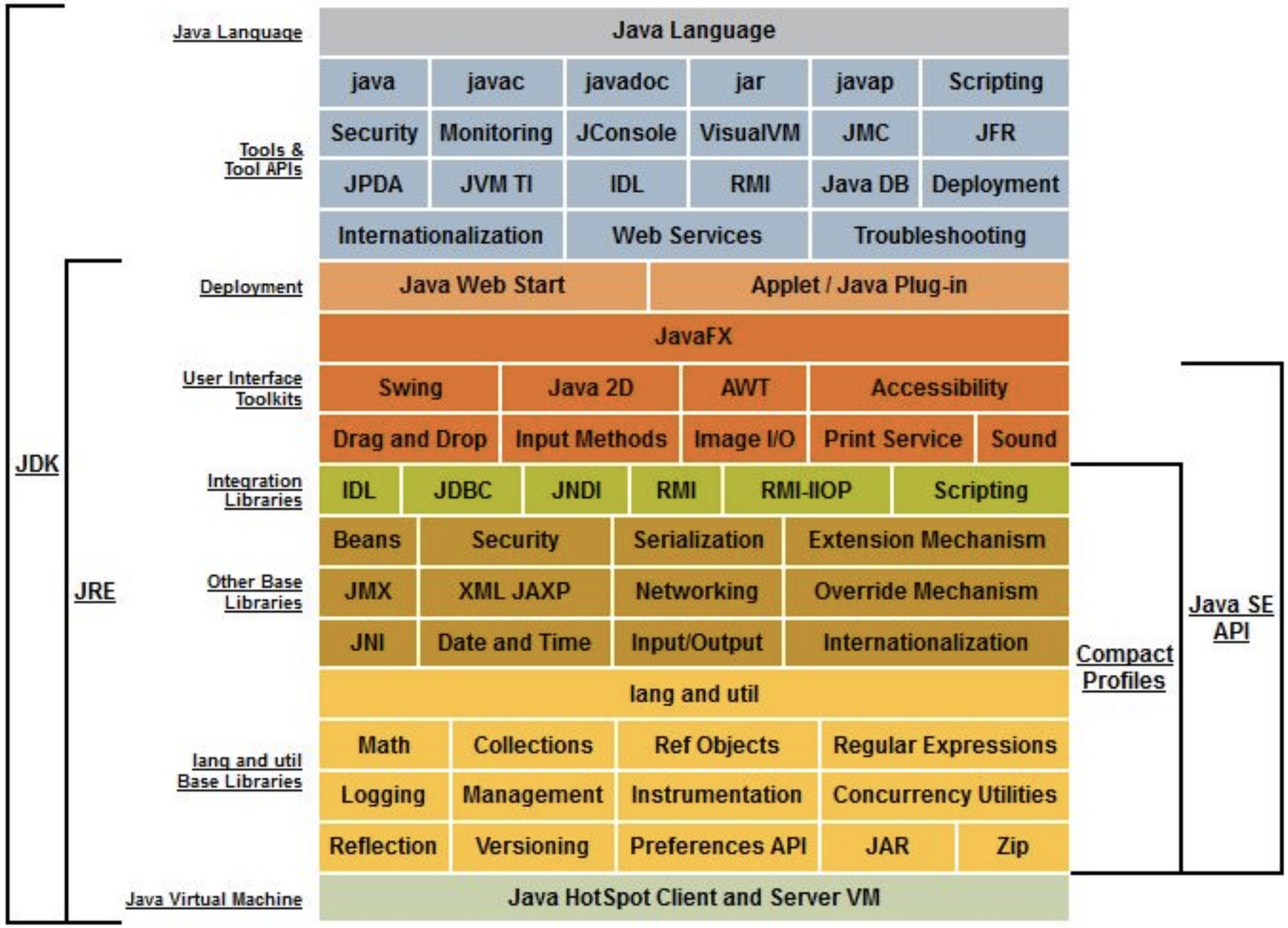
interprets bytecode into assembly



complies source into bytecode

JFK
(unrelated)

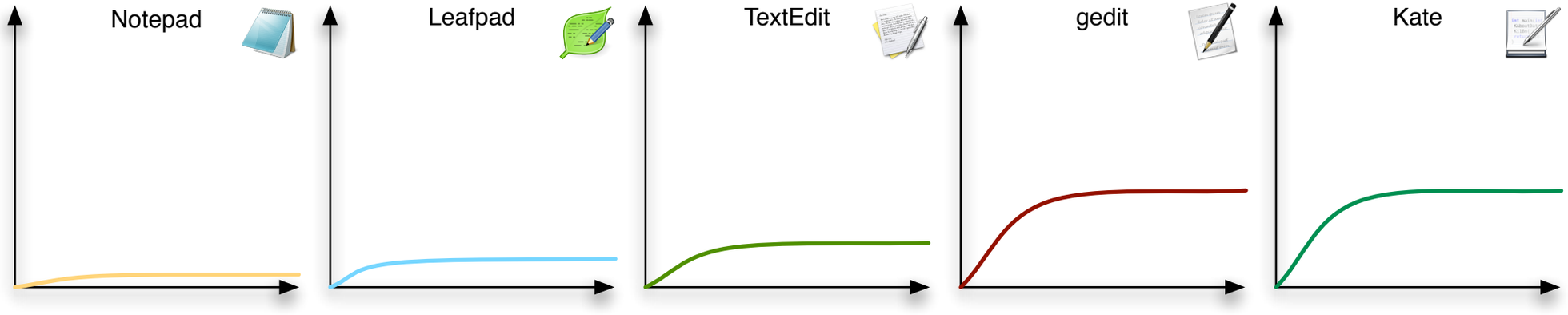
Is that all?



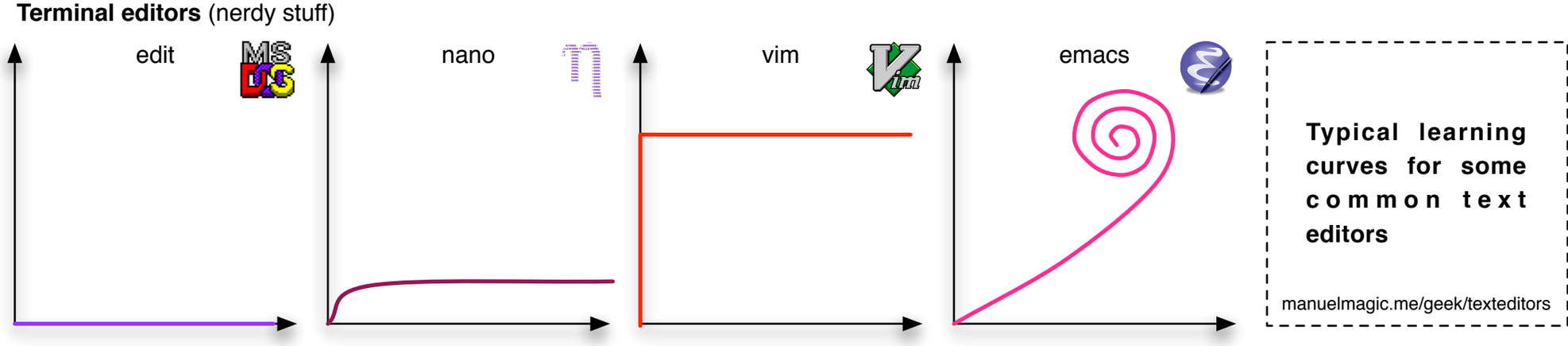
http://docs.oracle.com/javase/8/docs/index.html

Editors

System default editors



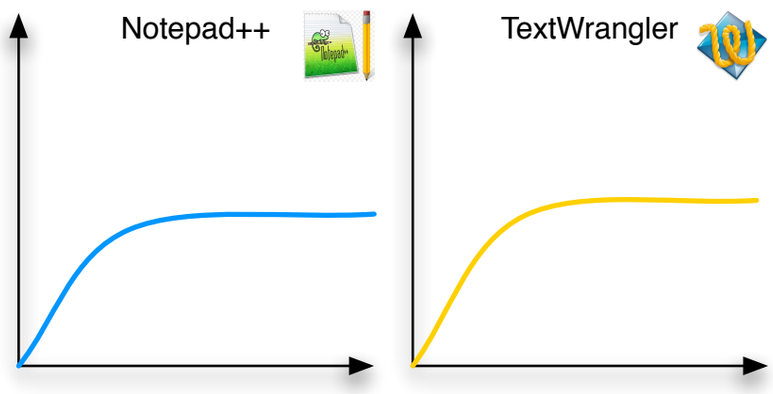
Terminal editors (nerdy stuff)



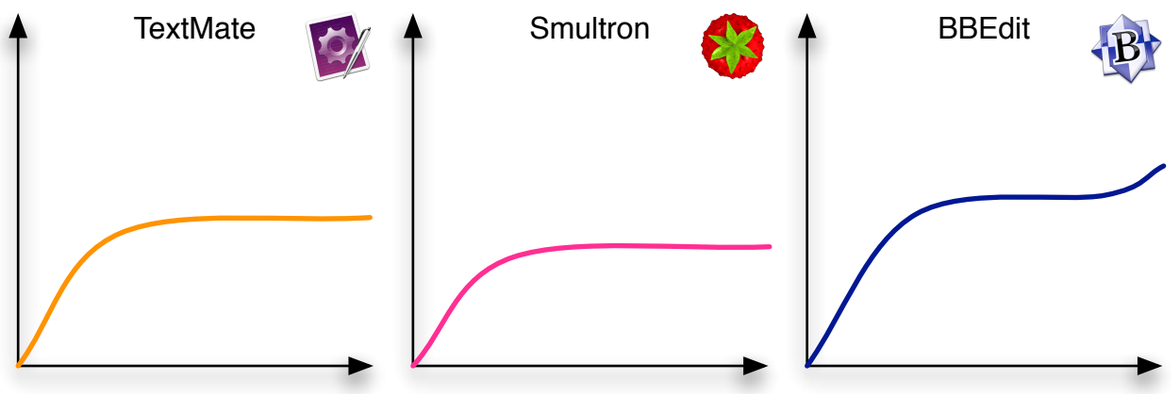
Typical learning curves for some common text editors

manuelmagic.me/geek/texteditors

Freeware editors (geeky stuff)



Shareware editors



Quick poll: who uses what?

The `main` Method

Classes as Programs

- Each class can have its own main method
- Consequently, you can horribly abuse classes and think of them as programs
- Remember that programming in main is not really object-oriented at all!
- In an ideal situation, you write one entry point for the whole program, but it's often handy for testing to have one at hand

Arguments

- Arguments from the command line can be passed into your program

```
java Prog arg0 arg1 arg2 …
```

- Arguments with spaces and special characters should be escaped with quotes

```
java Prog “(3 + 4) * 9)”
```

- Arguments come in as a String, so might need some parsing to be useful in other parts of the program

Hello, COMS10001!

- Write a program that uses arguments
- The Hello, COMS10001 program

Parsing Numbers

Parsing Numbers

- Arguments come in as String, so need to be handled specially if we expect them to become numbers

- There are several options available:

```
int x   = Integer.parseInt(“42”);

float f = Float.parseFloat(“0.027”);
```

Parsing Numbers

- Arguments come in as String, so need to be handled specially if we expect them to become numbers

- There are several options available:

```
int x   = Integer.parseInt(“42”);

float f = Float.parseFloat(“0.027”);
```

Can anyone spot the bug?

Parsing Numbers

- Unfortunately, float and double aren't arbitrarily precise, they only store *binary fractions*: ½ + ¼ + ⅛ …

```
float f = Float.parseFloat("0.027");
```

- Some numbers simply cannot be expressed

- Use BigDecimal instead

```
BigDecimal f = new BigDecimal("0.027");
```

Simple Calculator

- Write a program that uses arguments
- This time, add some numbers
- First, make sure you parse them!

Simple Calculator

```
class Adder {
  public static void main (String[] args) {
    int sum = 0;
      for (String arg : args) {
        sum = sum + Integer.parseInt(arg);
      }
      System.out.println("sum:" + sum);
  }
}
```

Exceptions

Exceptions

- Sometimes, things fail, for example, parsing!
- A try-catch block to handles exceptions

```
try {
  n = Integer.parseInt(s);
}
catch (Exception e){
  System.out.println("Bad string");
}
```

Exceptional Calculator

- Write a program that uses arguments
- This time, add some numbers
- In addition, print a message when things fail

Exceptional Calculator

```
class Adder {
  public static void main (String[] args) {
    int sum = 0;
    try {
      for (String arg : args) {
        sum = sum + Integer.parseInt(arg);
      }
      System.out.println("sum:" + sum);
    } catch (Exception e) {
      System.err.println("I don't know how to parse that!");
    }
  }
}
```

Output

Output

- Most of the time, your code will output to the stdout pipe

```
System.out.putstrln(“Hurrah, it worked!”);
```

- It’s good manners to output errors from your program to the stderr pipe instead

```
System.out.putstrln(“Boo, it broke!”);
```

- In your practicals, you should definitely do this: we use automarkers that look at stdout

Exit Codes

- When your program finishes, it always terminates with an exit code
- The code indicates to the operating system whether or not things went smoothly
- By default, the exit code is 0 when everything is fine: you can force your program to halt with this code

```
System.exit(0);
```

- All the other values indicate an error: you get to choose what they mean (between 1-127)

```
System.exit(42);  // error 42!
```

Simple Calculator

- Write a program that uses arguments
- This time, add some numbers
- In addition, print a message when things fail
- Tell the system that something broke!

Strings

Strings

- Strings in Java are a bit special
- You check for string equality with equals()

```
“foo”.equals(“bar”) // false
```

- Using == won’t quite do what you think (more on this later)

Comparing Strings

- Let's play with some strings, and compare them in various ways

Comparing Strings

```
class Equals {
  public static void main (String[] args) {
    System.out.println(3 == 4);             // false
    System.out.println("foo" == "bar");  // false
    System.out.println("foo" == "foo");  // true
    String foo1 = new String("foo");
    String foo2 = new String("foo");
    System.out.println(foo1 == foo2);       // false
    System.out.println(foo1.equals(foo2)); // true
    foo2 = foo1;
    System.out.println(foo1 == foo2);       // true
  }
}
```

Documentation

Documentation

- There are tons of functions out there to use
- Your best friend is the Java documentation
- Just Google “java 8 docs”

http://docs.oracle.com/javase/8/docs/

- There is ***a lot*** of off-the-shelf code that works across platforms

`toUpperCase` `substring` `matches` `replace`

Staying in C-World

Staying in C-World

- You can pretend you're not in Java and have nothing to do with objects
- To do that use the static keyword in front of your ~~methods~~ functions and ~~attributes~~ variables

Staying in C-World

- You can pretend you're not in Java and have nothing to do with objects
- To do that use the static keyword in front of your ~~methods~~ functions and ~~attributes~~ variables
- If you do this, you demonstrate very nicely that you've *completely* missed the point of Java and object-orientation

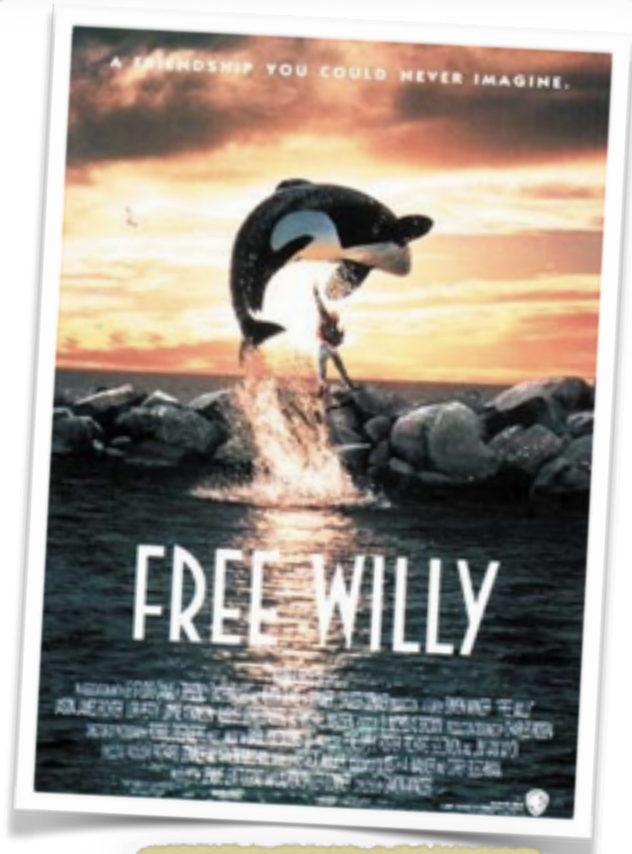
Staying in C-World

```
class CWorld {
  static int x;
  static int y;
  static int fib(int n) {
    switch (n) {
      case 0:
        return 1;
      case 1:
        return 1;
      default:
        return fib (n - 1) + fib (n - 2);
    }
  }

  public static void main (String[] args) {
    x = 0;
    y = 10;
    for (int i=x; i < y; i = i+1) {
      System.out.println(fib(i));
    }
  }
}
```

Staying in C-World

```
class CWorld {
  static int x;
  static int y;
  static int fib(int n) {
    switch (n) {
      case 0:
        return 1;
      case 1:
        return 1;
      default:
        return fib (n - 1) + fib (n - 2);
    }
  }

  public static void main (String[] args) {
    x = 0;
    y = 10;
    for (int i=x; i < y; i = i+1) {
      System.out.println(fib(i));
    }
  }
}
```

Please, don't do this: you need to be freed from C-World!



(it was big in 1993)