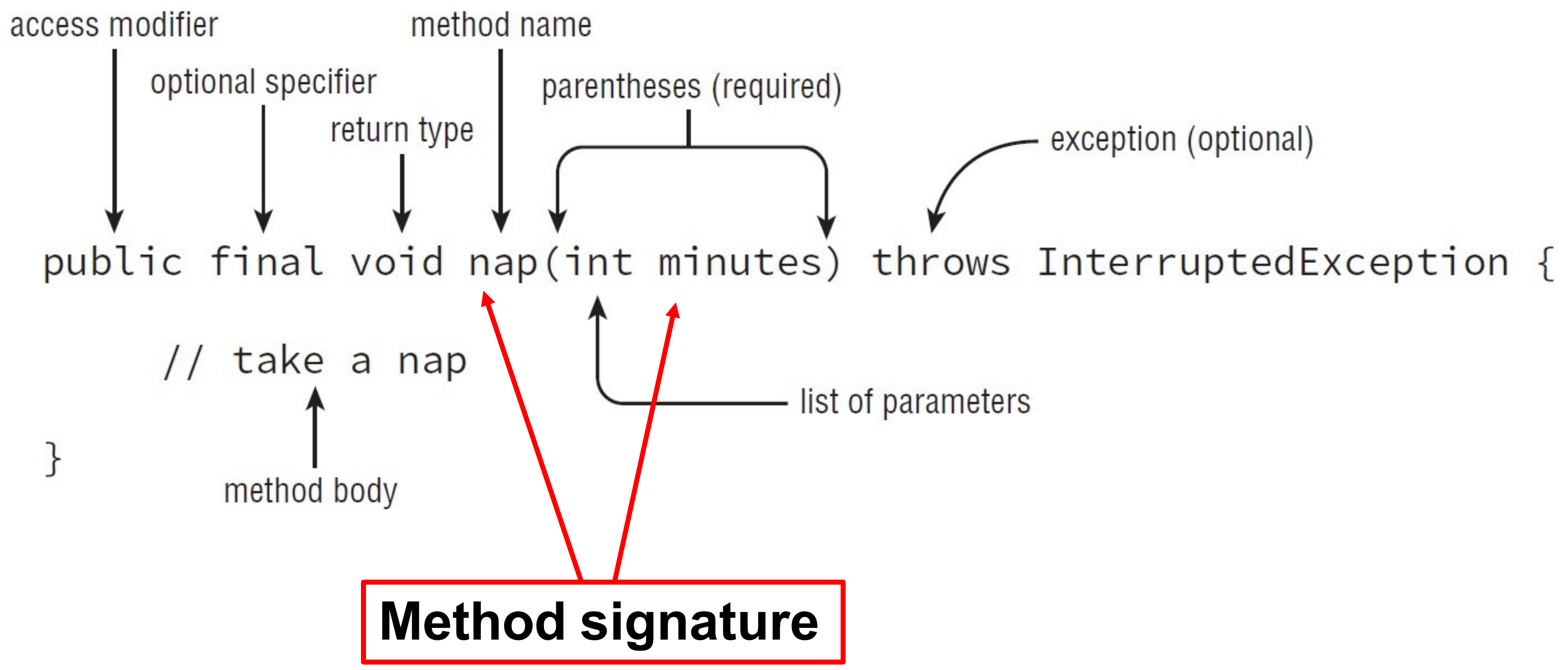
Methods

Ionut
Spalatelu

Method anatomy



```
public final void nap(int minutes) throws InterruptedException {
     // take a nap
}
```

Access

Access modifiers – give the method visibility outside the class:

- **public**: visible everywhere the class is visible
- **protected**: visible inside the package and in subclasses
- **[package-private]** or default: visible inside the package
- **private**: visible only inside the class

Optional specifiers

Optional specifiers:

- **static:** type of context (either static or non-static), or, in other words the method type (instance or static method)
- **final:** cannot override the method
- **syncronized**: allow only one running thread at a time
- **abstract:** method without a body (implementation)
- **default** (only allowed in interfaces): method with implementation inside an interface

Return type

Return type – the data type of the returned value:

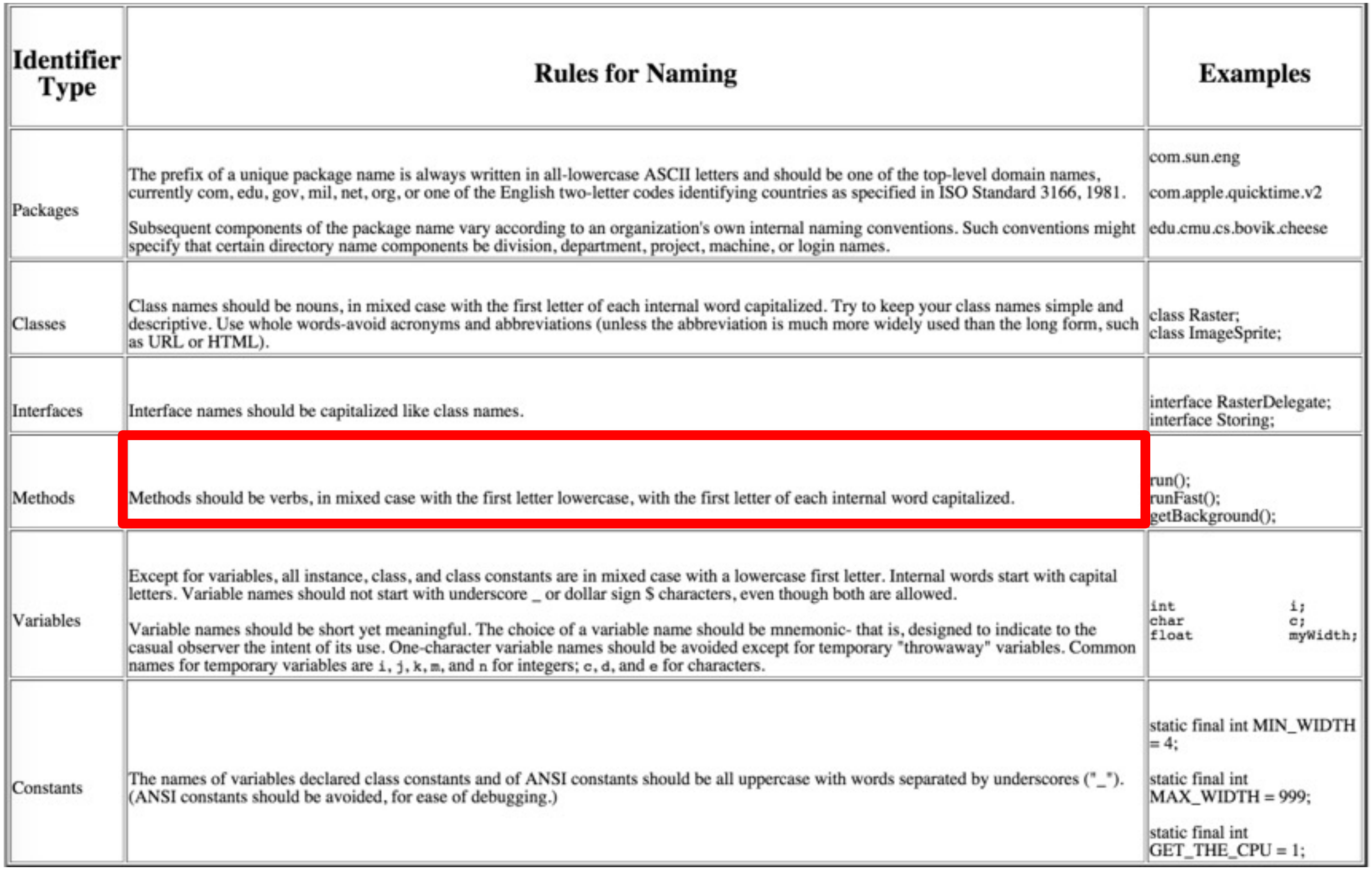
- is either **void,** if nothing is returned, or any valid Java type (either primitive or class type)

Method naming

Method name:

- needs to respect the Java **naming rules** (mandatory) Names in Java are allowed to contain letters, numbers, _ and **$** signs, and allowed to start with letters, or _ and **$** signs
- needs to respect the **naming conventions** (recommended):

Naming rules



Identifier Type	Rules for Naming	Examples
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese

Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	class Raster;
		class ImageSprite;

Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate;
		interface Storing;

Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	run();
		runFast();
		getBackground();

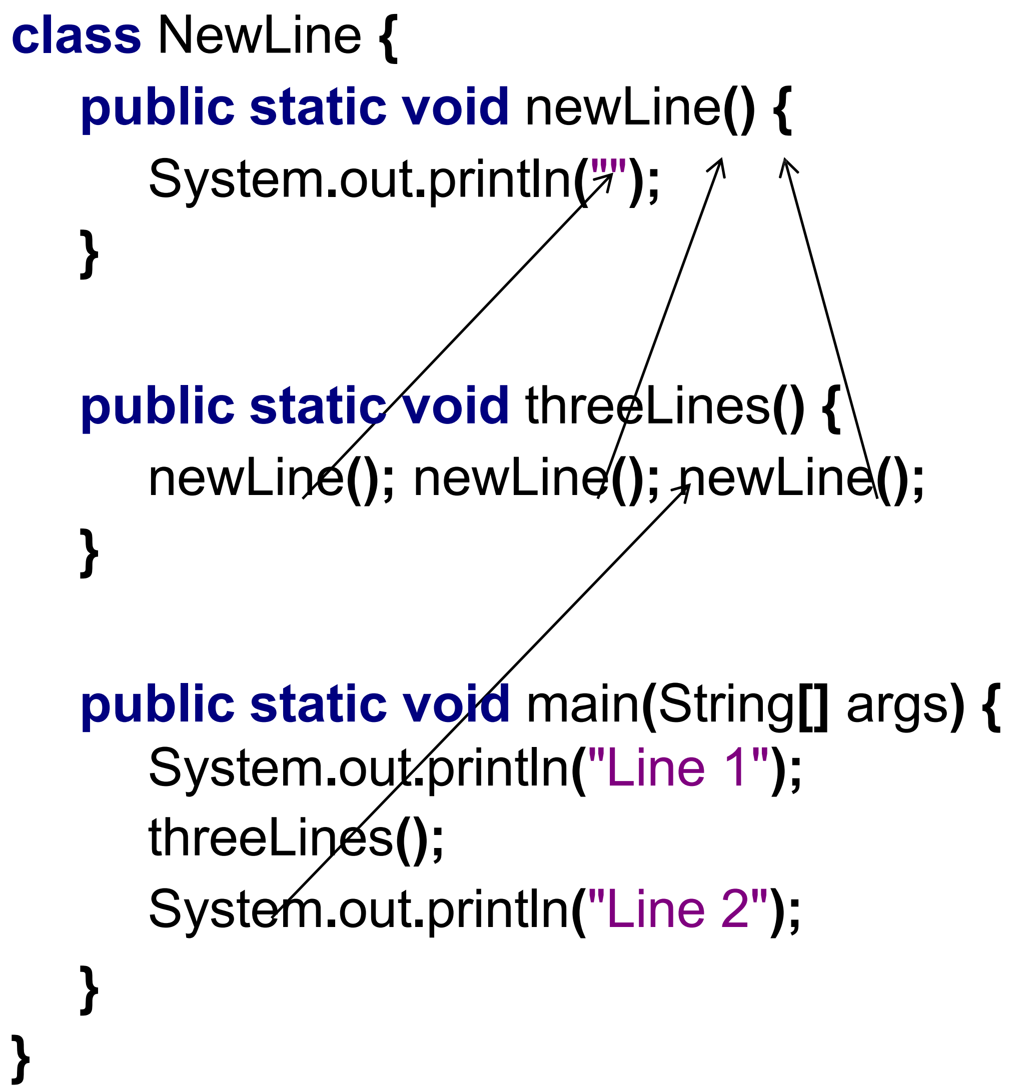
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.
	
	Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are `i`, `j`, `k`, `m`, and `n` for integers; `c`, `d`, and `e` for characters.
		`int i;`
		`char c;`
		`float myWidth;`

Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4;
		
		static final int MAX_WIDTH = 999;
		
		static final int GET_THE_CPU = 1;

Parameters

Parameters list:

- parentheses are always mandatory, even tough no parameter is present
- the number of parameters can be as big as needed
- they always need to have their **type** specified
- for var-args, the syntax is: Type... paramName
- If a simple parameter and a var-arg of the same type exist on the same method, the var-arg needs to be the last one.



```
class NewLine {
    public static void newLine() {
        System.out.println("");
    }

    public static void threeLines() {
        newLine(); newLine(); newLine();
    }

    public static void main(String[] args) {
        System.out.println("Line 1");
        threeLines();
        System.out.println("Line 2");
    }
}
```

```
class Square {
    public static void printSquare(int x) {
        System.out.println(x*x);
    }

    public static void main(String[] args) {
        int value = 2;
        printSquare(value);
        printSquare(3);
        printSquare(value*2);
    }
}
```

```
class Square2 {
    public static void printSquare(int x) {
        System.out.println(x*x);
    }

    public static void main(String[] args) {
        printSquare("hello");
        printSquare(5.5);
    }
}
```

What’s wrong here?

```
class Square3 {
    public static void printSquare(double x) {
        System.out.println(x*x);
    }

    public static void main(String[] args) {
        printSquare(5);
    }
}
```

What's wrong here?

Advantages of methods

- Big programs are built out of small methods
- Methods can be individually developed, tested and reused
- User of method (client) does not need to know how it works, only what it does

Mathematical Functions

```
Math.sin(x)

Math.cos(Math.PI / 2)

Math.pow(2, 3)

Math.log(x + y)

Math.floor(5.6)
```

Method overloading

- A feature that allows us to have more than one method with the **same name**, as long as we use different arguments or parameters
- It's the ability to create multiple methods of the **same name** with **different** implementations
- Calls to an overloaded method will run a specific implementation or version of that method
- **println** is good example of method overloading

Summary

- How to define methods and evaluate their visibility
- Var-args
- Method overloading

Questions

Bibliography

- https://docs.oracle.com/javase/tutorial/java/concepts/
- **Thinking in Java 4th Edition**, by Bruce Eckel
- http://beginnersbook.com/2013/04/oops-concepts/
- https://introcs.cs.princeton.edu/java/home/
- https://ocw.mit.edu/