

Programming Idioms

Eduardo Figueiredo

<http://www.dcc.ufmg.br/~figueiredo>

Programming Idioms

- Idioms are low-level patterns specific to a programming language
- An idiom describes how to solve a implementation-specific problem in a programming language
- Idioms ease communication among developers
 - They also speed up development and maintenance tasks

Idioms and Style

- A collection of related idioms defines a programming style
 - A programming style is characterized by the way language constructs are used
- Examples of idioms
 - The kind of loop statement used
 - Naming of program elements
 - Formatting of the source code

Recommendations

- Idioms are usually recommendations about the better use of language constructs
- For instance, some idioms show how to implement a design pattern in a specific programming language
 - How to implement Adapter in Java

A Catalog of Java Idioms

Eduardo Figueiredo

<http://www.dcc.ufmg.br/~figueiredo>

One Class per File

- To declare a single class per Java file
 - The unique class of the file should be declared public 1

Example

File Car.java



```
public class Car {
    String color;
    int velocity;

    void accelerate() {}
    void breaker() {}
}
```

The Main Method

- To place the *main()* method in a separate class, apart from business logic 2
- 3 ○ Only initialization code should be implemented in a class holding *main()*
- Example

```
public class CarTest {
    public static void main(String[] args) {
        ...
    }
}
```

Hiding Attributes

- Attributes should be private or protected 4
- 5 ○ Getting and setting methods should be used to access the class data

```
public class Car {
    private String color;
    protected int velocity= 0;
    public void accelerate() {...}
    public void breaker() {...}
    public void setColor(String newColor)
    { color = newColor; }
    public String getColor() { return color; }
}
```

Example: Car

```
public class Car {
    private String color;
    protected int velocity = 0;
    public void accelerate() {
        velocity++;
    }
    public void breaker() {
        velocity--;
    }
    public void setColor(String newColor)
    { color = newColor; }
    public String getColor() { return color; }
    public String toString()
    { return "Car " + color + " : " + velocity; }
}
```

```
public class CarTest {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.setColor("Black");
        System.out.println(myCar);
    }
}
```

CarTest.java

Car.java

Naming Conventions

- To use “*camel case*” in names of classes, methods, and attributes
- 6 ○ Class names should be a noun and begin with upper case
- 7 ○ Method names should be a verb and begin with lower case
- 8 ○ Attribute names should be a noun or adjective and begin with lower case

Indentation and Comments

- To indent nested structure 9
- To separate with a blank line a comment from the previous code 10
- Comments should refer to the block of code that follows them 11

Example: Car2

```
public class Car2 {
    public static final int LIMIT = 150;
    protected int velocity = 0;
    public void accelerate() {
        velocity++;
    }
    // Test the car velocity limit.
    if (velocity > LIMIT) {
        System.out.println("Bib bib bib.");
        velocity = LIMIT;
    }
    public void breaker() {
        if (velocity > 0)
            velocity--;
    }
}
```

9

10

blank line

11

commented block

Decrease Variable Scope

- To define nested blocks to declare local variables with reduced scope 12
 - Nested blocks can also be used to make it clear the code a comment refers to

Example: Car3

```
public class Car3 {
    public static final int LIMIT = 150;
    protected int velocity = 0;
    public void accelerate() {
        velocity++;
        // Print alert.
        {
            String alert = "Bib.";
            if (velocity == LIMIT)
                System.out.println(alert);
        }
        // Test the car velocity limit.
        if (velocity > LIMIT)
            velocity = LIMIT;
        ...
    }
}
```

12
nested block

Variable Declarations

- To avoid naming a local variable with the same name of a field or class variable 13
 - Method names should also not clash with local variables and fields
- If possible, declare and initialize a field or local variable in the same command line 14

Expression

- To avoid a complex logical expression in a single *if* command 15
 - You should divide the complex expression in nested *if* commands
- To avoid the ternary operator “?” when the logic expression is complex 16
 - In this case, use the *if* command instead

```
public void breaker() {
    velocity = ( velocity < 0 ) ?
        ( velocity == MIN ? STOPPED :
          min(velocity) ) : decrease(velocity);
}
```

Switch Case

- To keep the code of each case in a *switch* short and simple 17
 - About 5 lines per case
 - Long code can be extracted to a new method
- To always finish a case with the *break* command 18

Default Option in a Switch

- To include the *default* option in *switch* commands 19
- The *default* option should only capture the unexpected situation 20
 - All expected situations should be implemented as cases in the *switch*

[Loops]

- Do not use temporary variables to finish a loop
 - You should use the *break* command to finish a loop before the exit condition be met
 - You should use the *return* command to immediately test the exit condition

21

[Empty Blocks and GOTO]

- All empty blocks { } should have a comment inside in order to explain the reasons for being empty
- You must not use *goto*

22

23

[Bibliography]

- F. Buschmann et al. **Pattern-Oriented Software Architecture: A System of Patterns**. John Wiley & Sons, 1996.
 - Chapter 4 Idioms
- A. von Staa. **Programação Modular**. Elsevier, 2000.
 - Appendix 3, 4 e 5