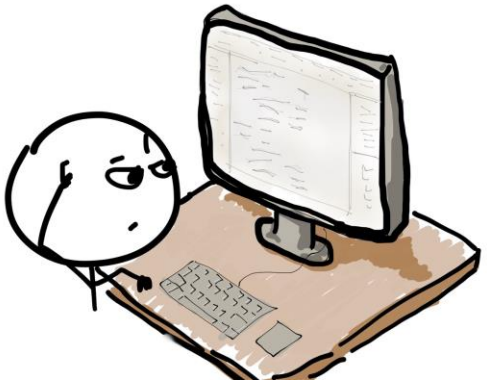
Programming Idioms

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 conventions
  - Formatting matters

Recommendations

- Idioms are usually recommendations about the better use of language constructs
- Some idioms show how to implement a design pattern in a specific programming language
  - For instance, 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 a 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
  - Only initialization code should be implemented in the “main” class
- Example

2

3

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

Hiding Attributes

- Attributes should be private or protected

4

  - Getting and setting methods should be used to access the class data

5

```
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; }
}
```

Car.java

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

CarTest.java

Naming Conventions

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

6

7

8

Indentation and Comments

- You should…
  - … indent nested structure
  - … separate with a blank line a comment from the previous code
- Comments should refer to the block of code that follows them

9

10

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

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

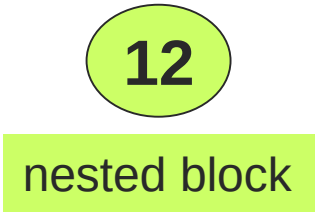
Example: Car3

```
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;
  }
 ...
}
```



Variable Declarations

- To avoid naming a local variable with the same name of a field or class variable
  - 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

13

14

Expression

- To avoid a complex logical expression in a single *if* command
  - You should divide the complex expression in nested *if* commands

15

- To avoid the ternary operator "?" when the logic expression is too complex
  - In this case, use the *if* command instead

16

```
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
  - About 5 lines per *case*
  - Long code can be extracted to a new method

17

- To always finish a *case* with the *break* command

18

Default Option in a *Switch*

- Always include the *default* option in *switch* commands

19

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

20

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 *continue* command to immediately test the exit condition

Empty Blocks

- All empty blocks { } should have a comment inside in order to explain the reasons for being empty

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