Mohammadmahdi Farrokhy

Clean Code:

A Handbook of Agile Software Craftsmanship

Chapter 02: Meaningful Names

Meaningful names characteristics:

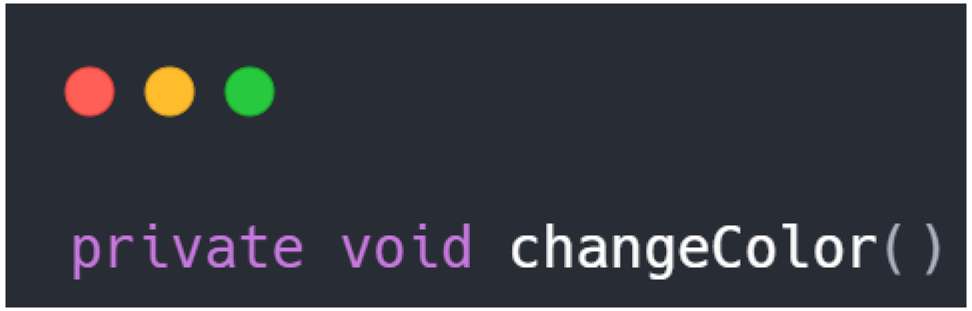
Name of the classes, methods and variables must have some characteristics to make it easier to understand the code. In this chapter we will learn a few of them.

Meaningful names characteristics:

1. The name should be intension-revealing.
   - What is the purpose behind this element of code? (method, object, variable, …)
   - What was the author of the code thinking about and what was he looking for?
   - What is this module supposed to do?

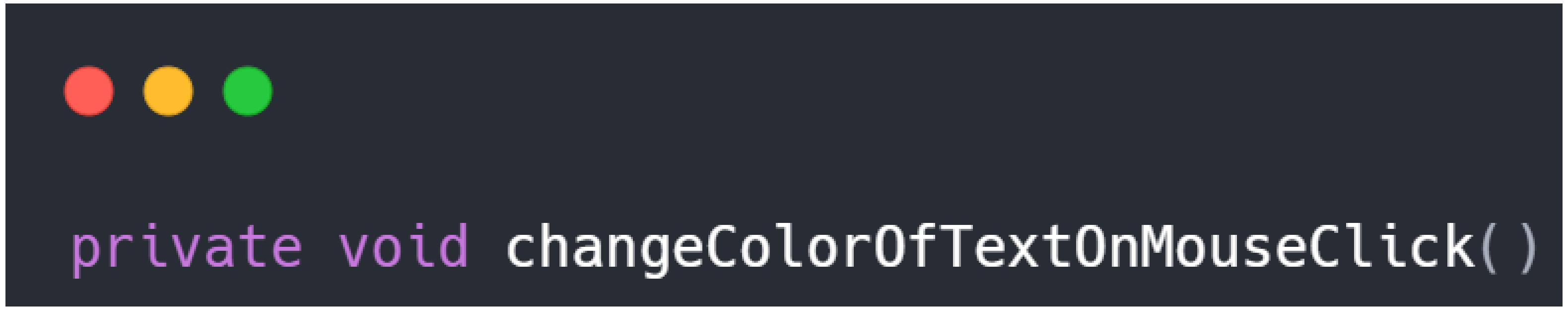
The name should be chosen in a way that no comments or documents are needed to answer these questions. Name of the element should answer all the big questions. If a name needs comment, it doesn't reveal its intent.

Meaningful names characteristics:



```
private void changeColor( )
```

✗



```
private void changeColorOfTextOnMouseClick( )
```

✓

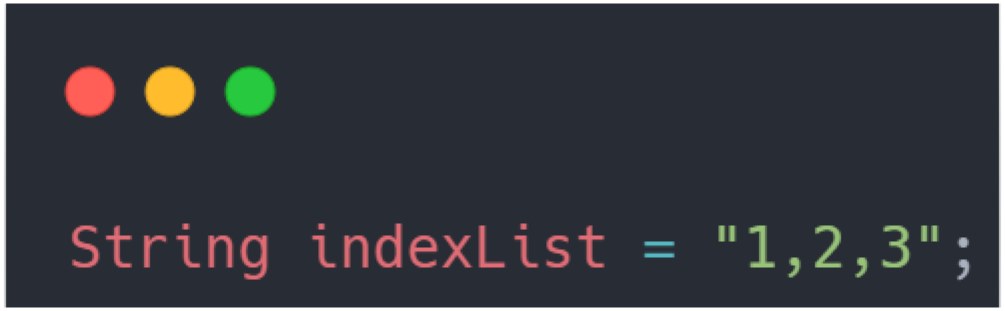
Meaningful names characteristics:

2. The name should not give dis-information.

   Using names that may mislead the reader, will take more time to understand the code. Or even take his mind towards another concept with a similar name, or hide the meaning of the code. Some possible situations:

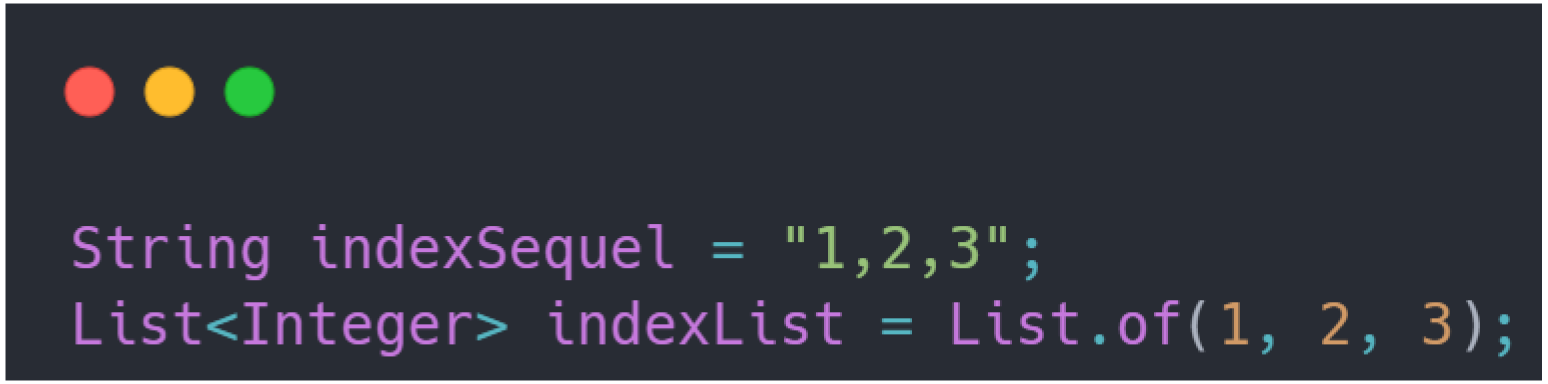
   - Using the name **hp** as the acronym of **hypotenuse.**
   - Using the name **accountList** while its type is not **List.** Instead we can use **accountGroup**, **bunchOfAccounts** or even **accounts.**
   - Using long similar names like **XYZControllerForEfficientHandlingOfStrings** and **XYZControllerForEfficientStorageOfStrings.**
   - Using the letters **O** and **l** along side with the numbers **0** and **1.**

Meaningful names characteristics:



```java
String indexList = "1,2,3";
```

✗



```java
String indexSequel = "1,2,3";
List<Integer> indexList = List.of(1, 2, 3);
```

✓

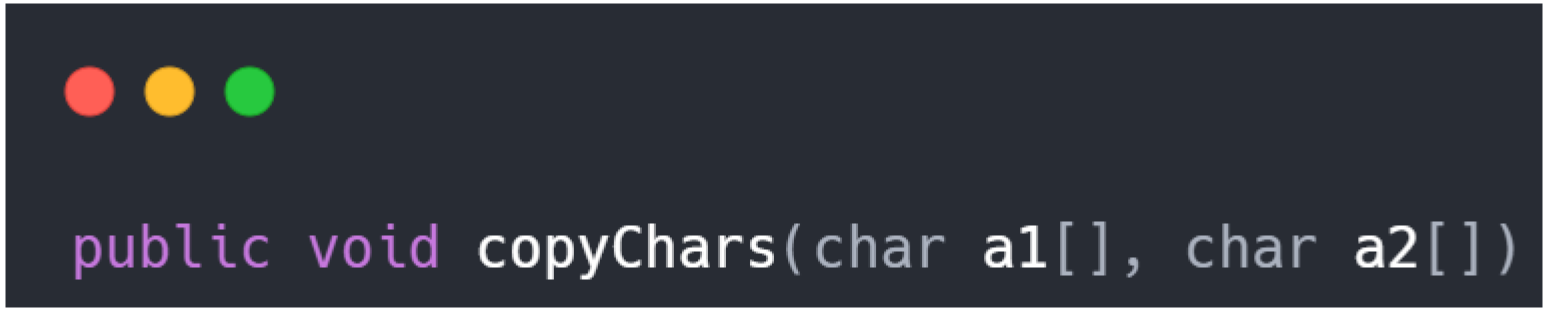
Meaningful names characteristics:

3. The name should make meaningful distinctions.

   We should pick a name for code elements in a way to differ them easily, no matter how much they are alike. Some examples:

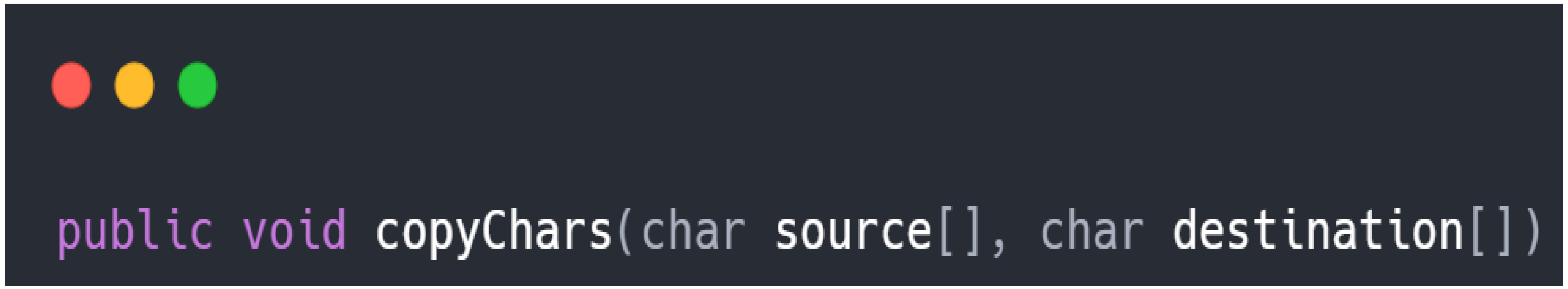
   - The names in a scope, or method arguments should not be singular letter and similar, like they are only to satisfy the compiler. The reader should not be forced to read the algorithm to understand the intention of the variable.
   - Three classes named **Product**, **ProductInfo** and **ProductData**. What is the intention behind each one?
   - Using words like **variable**, **method** or **class** for naming.

Meaningful names characteristics:



```
public void copyChars(char a1[], char a2[])
```

✗



```
public void copyChars(char source[], char destination[])
```

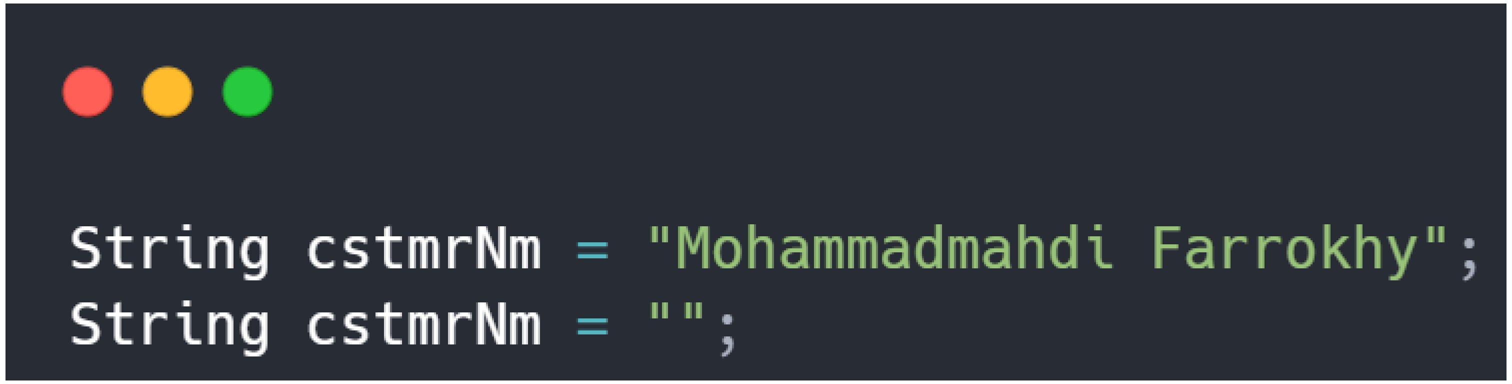
✓

Meaningful names characteristics:

4. The name should be pronounceable.

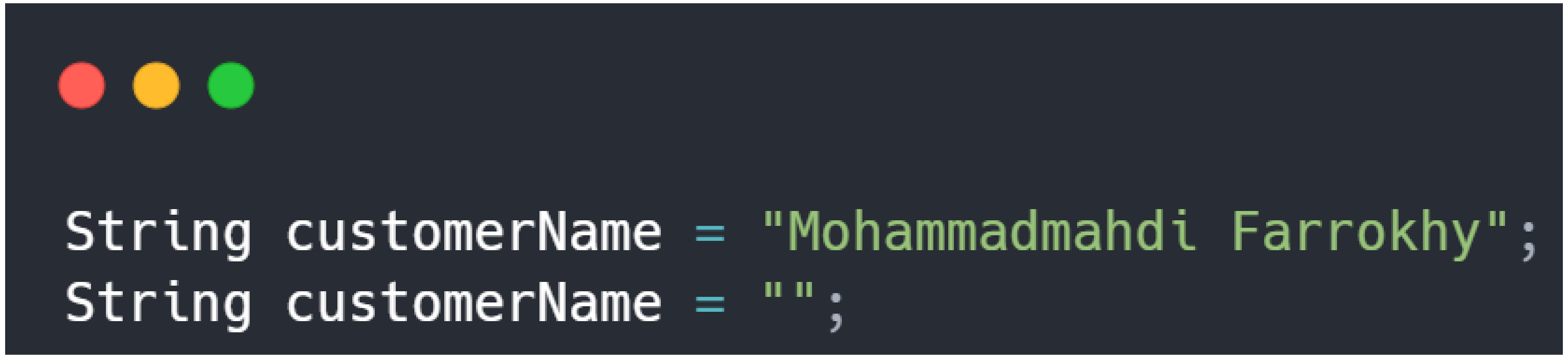
   Maybe using **bthymdh** as a variable name look like a good decision as the acronym of **birthdayYearMonthDayHour** in the first look. But every time you want to read this part of code you will be forced to wait for a few seconds so you could understand and read this variable.

Meaningful names characteristics:



```java
String cstmrNm = "Mohammadmahdi Farrokhy";
String cstmrNm = "";
```

✗



```java
String customerName = "Mohammadmahdi Farrokhy";
String customerName = "";
```

✓

Meaningful names characteristics:

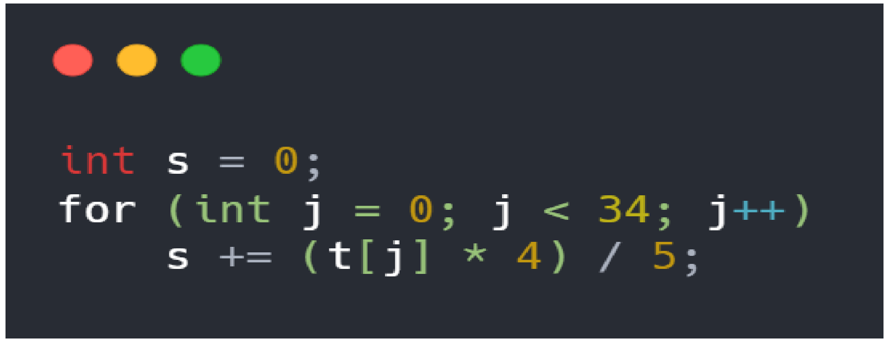
5. The name should be searchable.

   Using singular letter variables or constant numbers in the code will make it difficult to find them in code with the eye quickly.

   – Variable names should be more than 3 characters long.

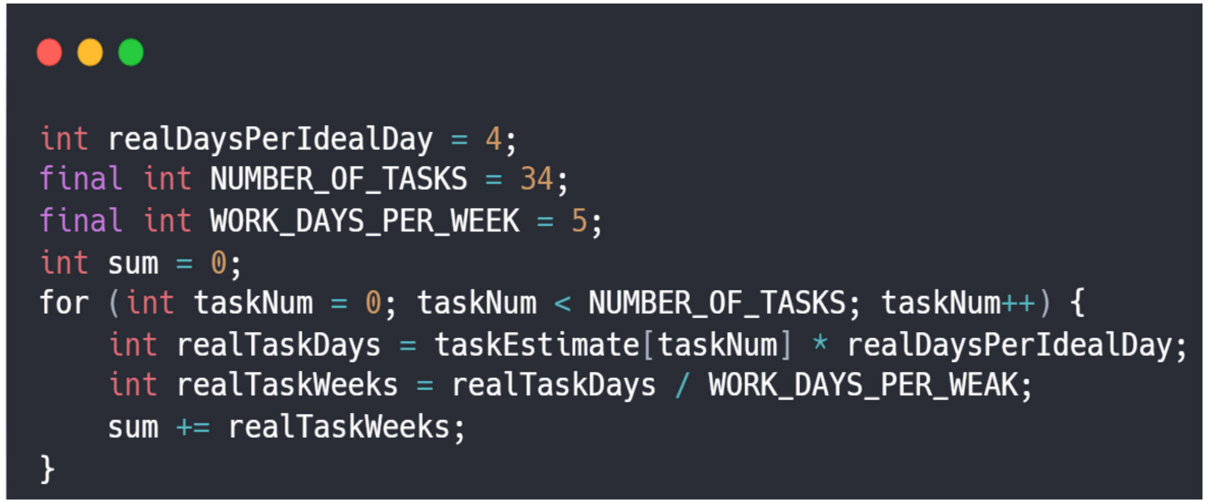
   – Constant numbers should be stored in a **final** variable and use the variable where needed.

Meaningful names characteristics:



```
int s = 0;
for (int j = 0; j < 34; j++)
    s += (t[j] * 4) / 5;
```

✗



```
int realDaysPerIdealDay = 4;
final int NUMBER_OF_TASKS = 34;
final int WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for (int taskNum = 0; taskNum < NUMBER_OF_TASKS; taskNum++) {
    int realTaskDays = taskEstimate[taskNum] * realDaysPerIdealDay;
    int realTaskWeeks = realTaskDays / WORK_DAYS_PER_WEAK;
    sum += realTaskWeeks;
}
```

✓

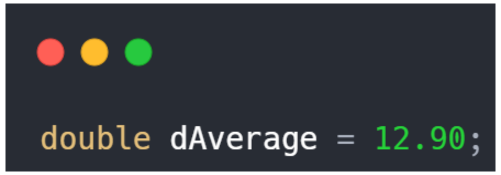
Meaningful names characteristics:

6. The name should avoid encoding.

   The code is complex by itself. So we don't need extra complexity caused by encoded names. Suh as:

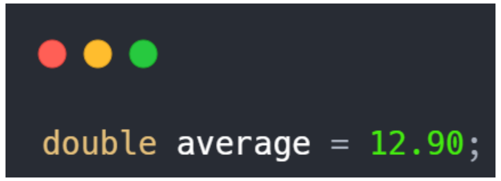
   - Names with member prefix: **m_** is a sign of class members used in old IDEs. Today this kind of names are obsolete.
   - Hungarian notation: It was used in the old IDEs where there was a distance between object declaration and its usage; and the programmer wanted to remember the object type.

Meaningful names characteristics:



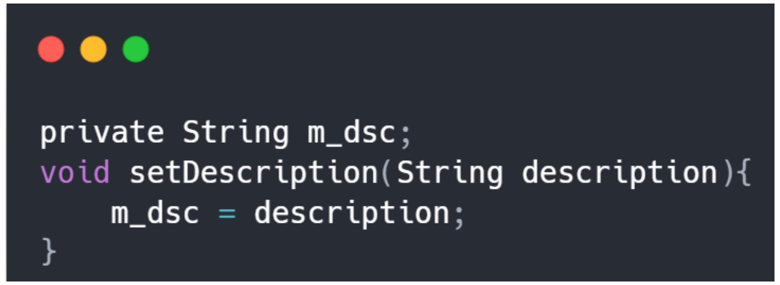
```java
double dAverage = 12.90;
```

✗



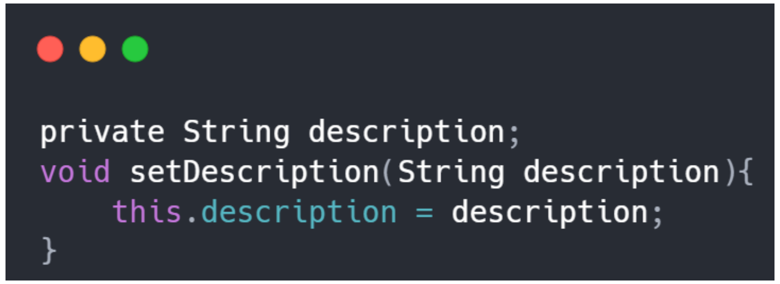
```java
double average = 12.90;
```

✓



```java
private String m_dsc;
void setDescription(String description){
    m_dsc = description;
}
```

✗



```java
private String description;
void setDescription(String description){
    this.description = description;
}
```

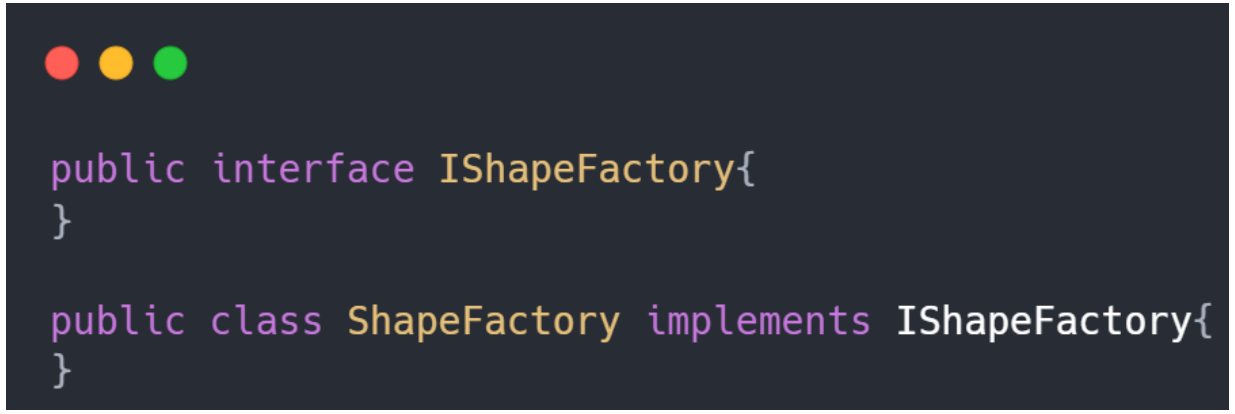
✓

Meaningful names characteristics:

7. The name of **interfaces** and their implementor **classes**.

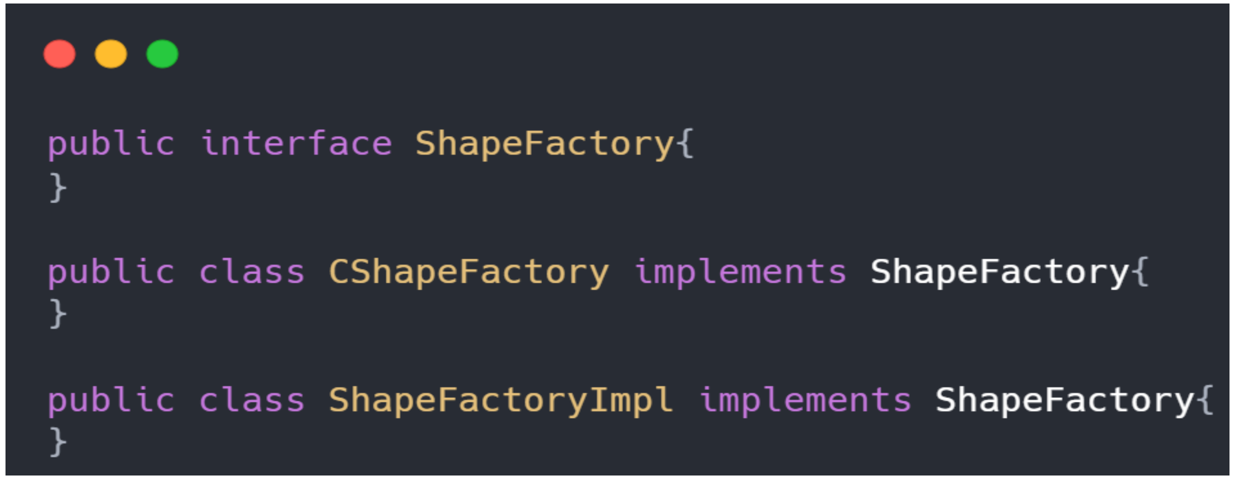
   Don't use **I** at the beginning of an **interface** name. Instead write a **C** at the beginning or **Impl** at the end of its implementor classes.

Meaningful names characteristics:



```java
public interface IShapeFactory{
}

public class ShapeFactory implements IShapeFactory{
}
```

✗



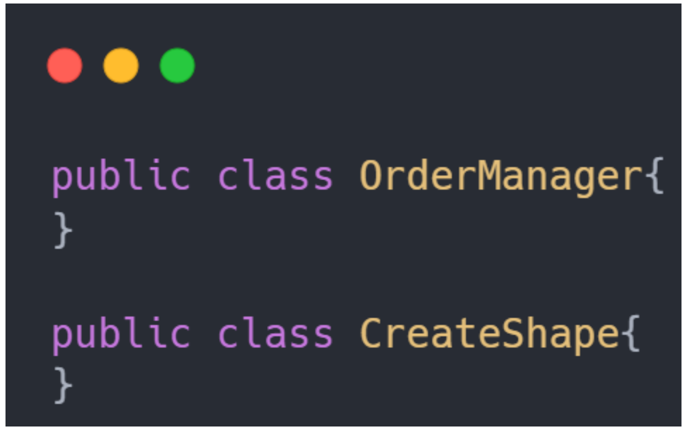
```java
public interface ShapeFactory{
}

public class CShapeFactory implements ShapeFactory{
}

public class ShapeFactoryImpl implements ShapeFactory{
}
```

✓

Meaningful names characteristics:

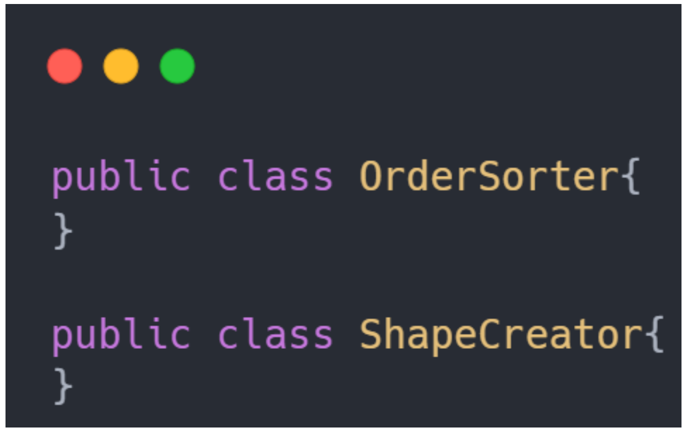
8. The name of **classes:**
   - It should be noun or noun phrase like Customer, Account and AddressParser.
   - It should not contain general words like Manager, Processor, Data or Info that indicate a big variety of tasks and do not reveal the true functionality of the class.

Meaningful names characteristics:



```java
public class OrderManager{
}

public class CreateShape{
}
```

✗



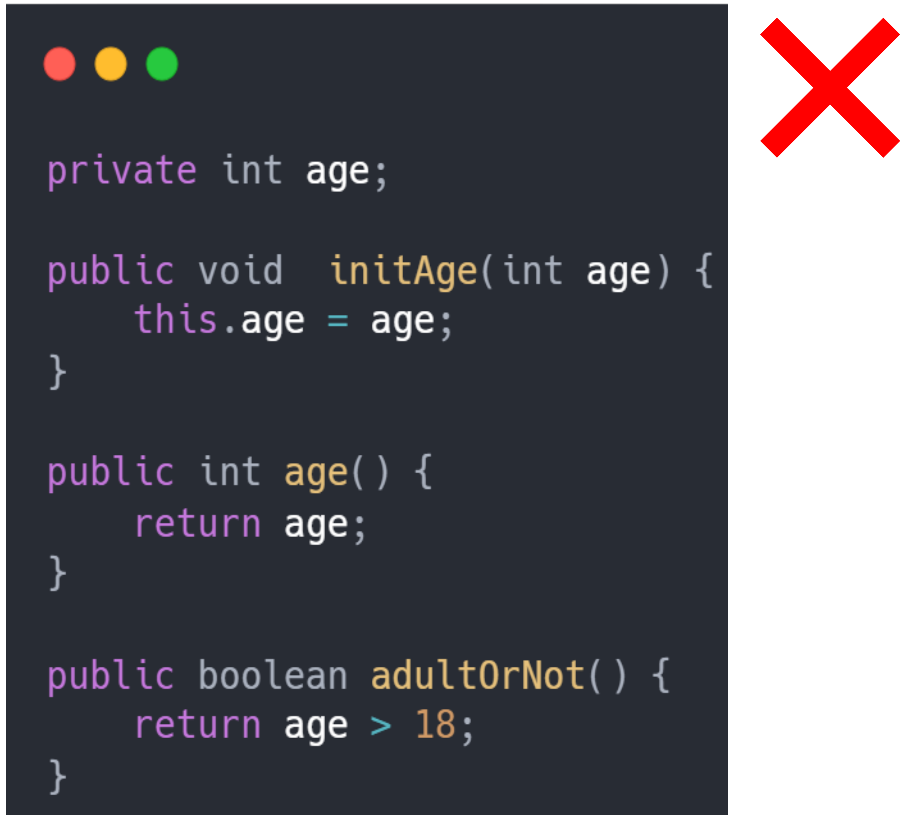
```java
public class OrderSorter{
}

public class ShapeCreator{
}
```

✓

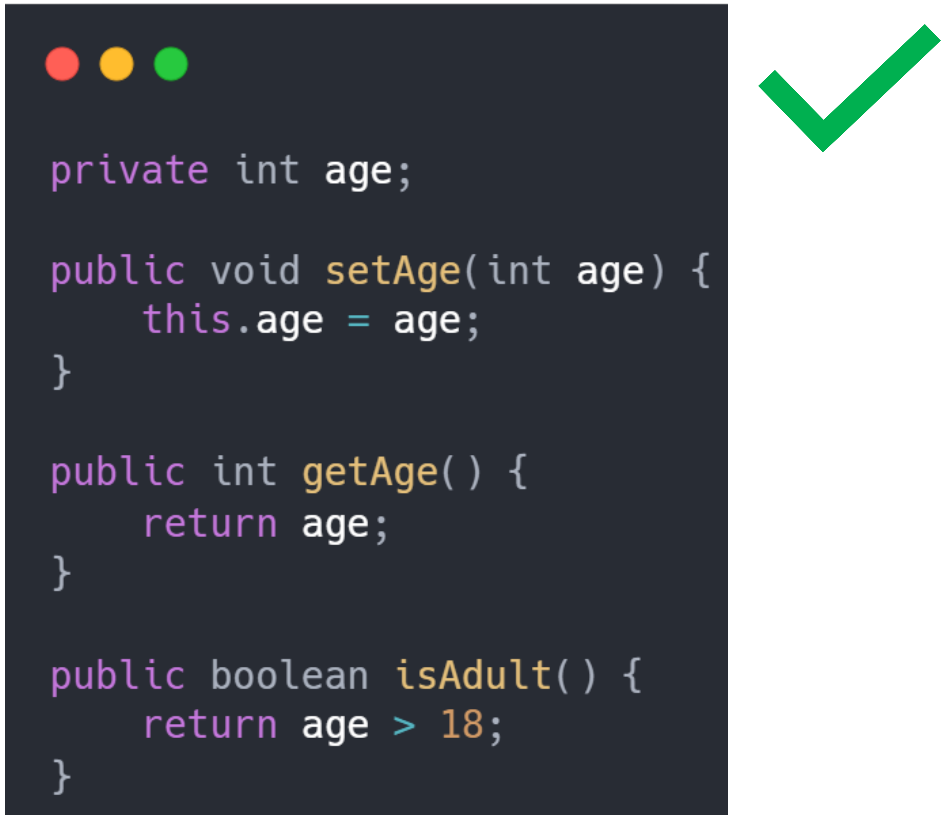
Meaningful names characteristics:

9. The name of **methods**:
   - It should be verb or verb phrase like deletePage, calculateDistance and postPayment.
   - **Setter** methods should start with **set**.
   - **Getter** methods should start with **get**.
   - Boolean **methods** should start with **is**.

Meaningful names characteristics:



```
private int age;

public void  initAge(int age) {
    this.age = age;
}

public int age() {
    return age;
}

public boolean adultOrNot() {
    return age > 18;
}
```



```
private int age;

public void setAge(int age) {
    this.age = age;
}

public int getAge() {
    return age;
}

public boolean isAdult() {
    return age > 18;
}
```

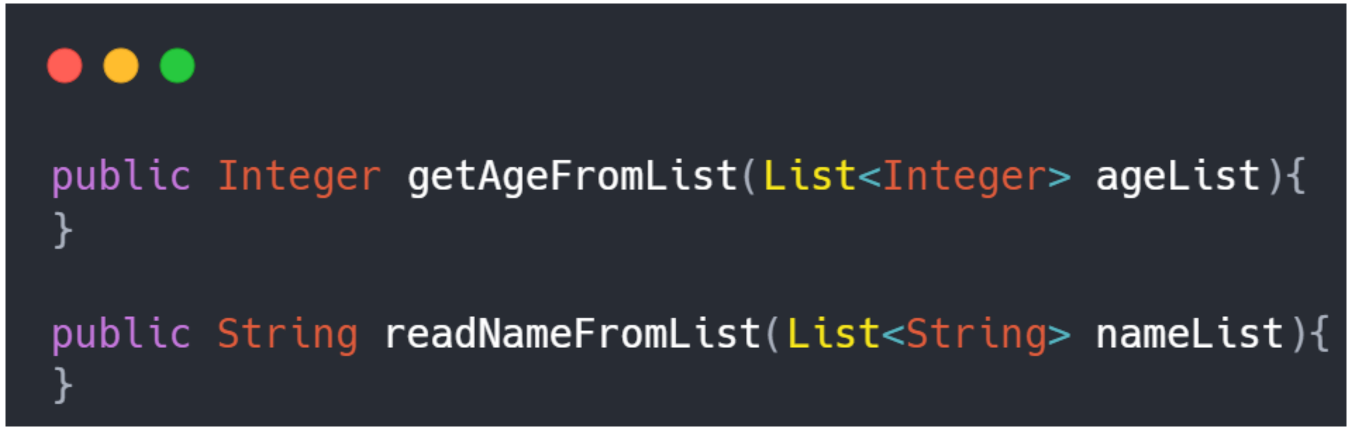
Meaningful names characteristics:

10. Use one word for one concept.

    If you once used **insert** as a keyword for adding a new element to a collection, use **insert** for any situation you add a new element to a collection.

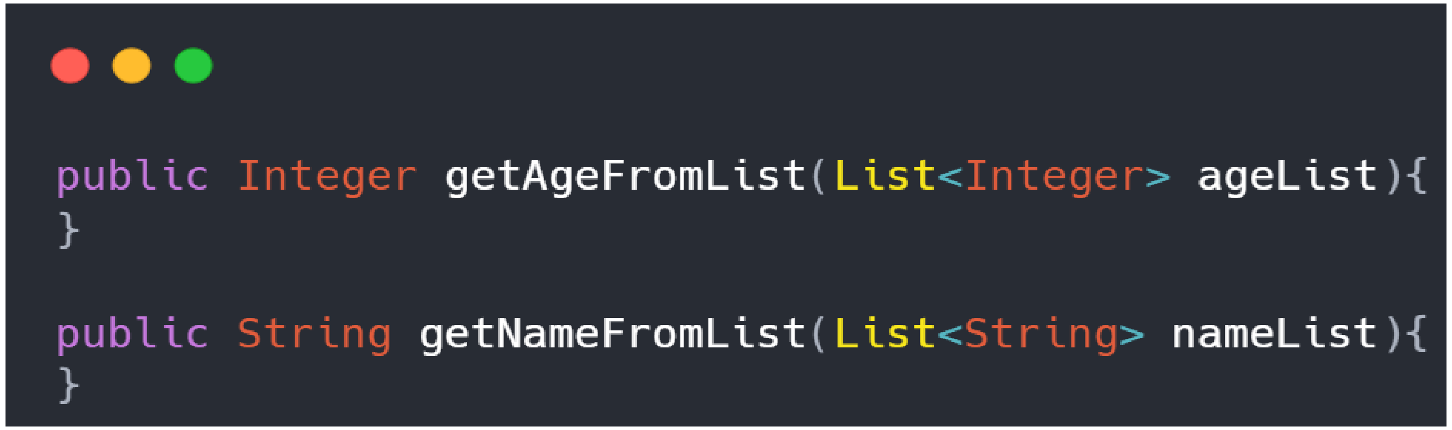
    If you once used **get** as a keyword for getting an existing element in a collection, use **get** for any situation you get an existing element in a collection.

Meaningful names characteristics:



```
public Integer getAgeFromList(List<Integer> ageList){
}

public String readNameFromList(List<String> nameList){
}
```

✗



```
public Integer getAgeFromList(List<Integer> ageList){
}

public String getNameFromList(List<String> nameList){
}
```

✓

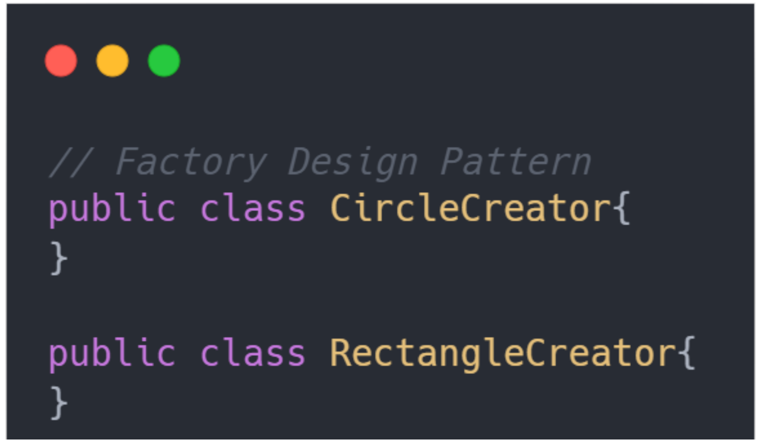
Meaningful names characteristics:

11. Use solution domain names.

    It is better to use computer science terms, design patterns and algorithms names instead of using the problem domain names all the time. Using problem domain names requires the presence of a domain expert.

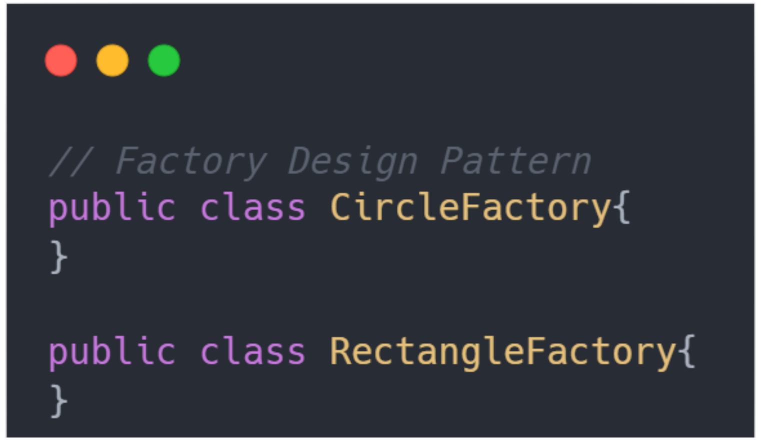
12. But if you have access to the domain expert, choosing problem domain names can be a good decision.

Meaningful names characteristics:



```java
// Factory Design Pattern
public class CircleCreator{
}

public class RectangleCreator{
}
```

✗



```java
// Factory Design Pattern
public class CircleFactory{
}

public class RectangleFactory{
}
```

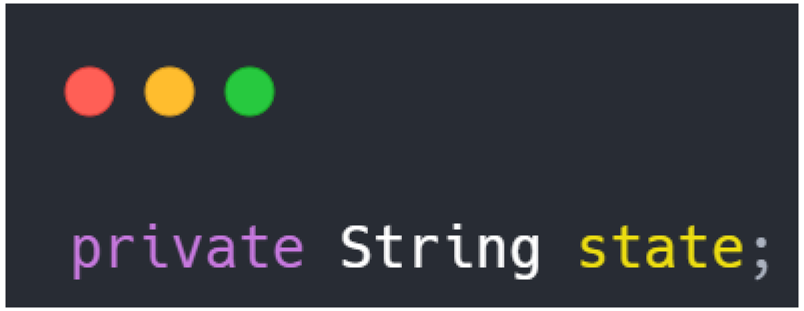
✓

Meaningful names characteristics:

13. Add meaningful context.

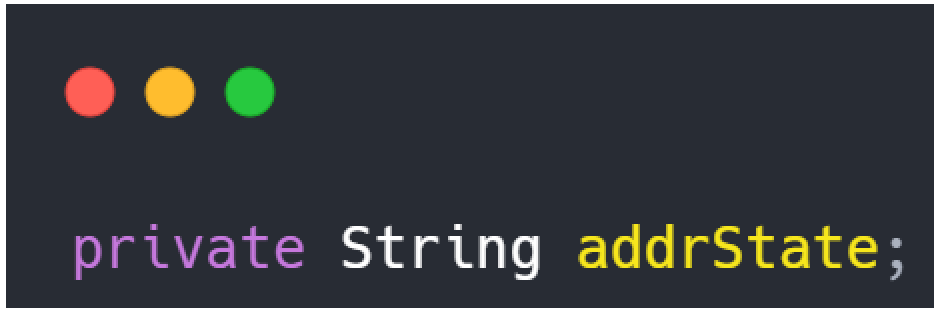
    Some names may be ambiguous by themselves. Adding a meaningful context can clear the naming.

Meaningful names characteristics:



```
private String state;
```

✗



```
private String addrState;
```

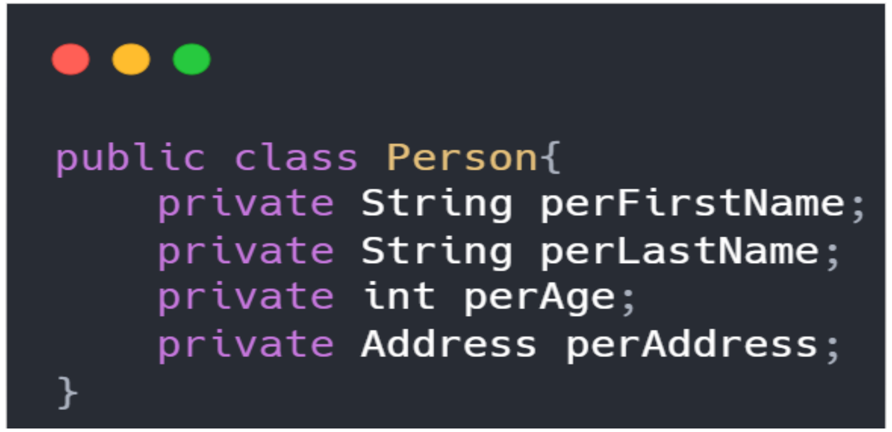
✓

Meaningful names characteristics:

14. Don't add gratuitous context.

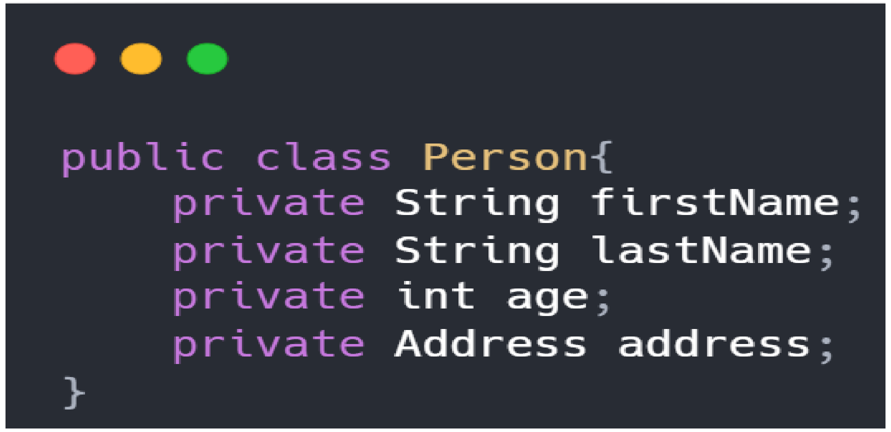
    Some prefixes or postfixes only add more complexity. Imagine adding **per** at the beginning of every property name in class **Person.** By writing **per** the IDE suggests too many different objects and it confuses us.

Meaningful names characteristics:



```
public class Person{
    private String perFirstName;
    private String perLastName;
    private int perAge;
    private Address perAddress;
}
```

✗



```
public class Person{
    private String firstName;
    private String lastName;
    private int age;
    private Address address;
}
```

✓

In brief meaningful name has these features:

- The name is not **dis-informative**.
- The name doesn't remind the reader of **another concept**.
- The name doesn't need **comments** to be understood.
- The name reveals the author's **intentions**.
- The names of the classes, objects and variables are **nouns** or **noun phrases**.
- The names of the methods are **verbs** or **verb phrases**.
- The names of the enums are **adjectives**.
- The names of the packages are **nouns** or **noun phrases**.
- There is a **semantic continuity** between name of the class with its methods and properties.
- There is no **similar** and **duplicated** names.
- The more the **distance** of a variable's declaration and usage, the **longer** its name.
- The more **public** a method, the more **general** its task and the **shorter** its name.