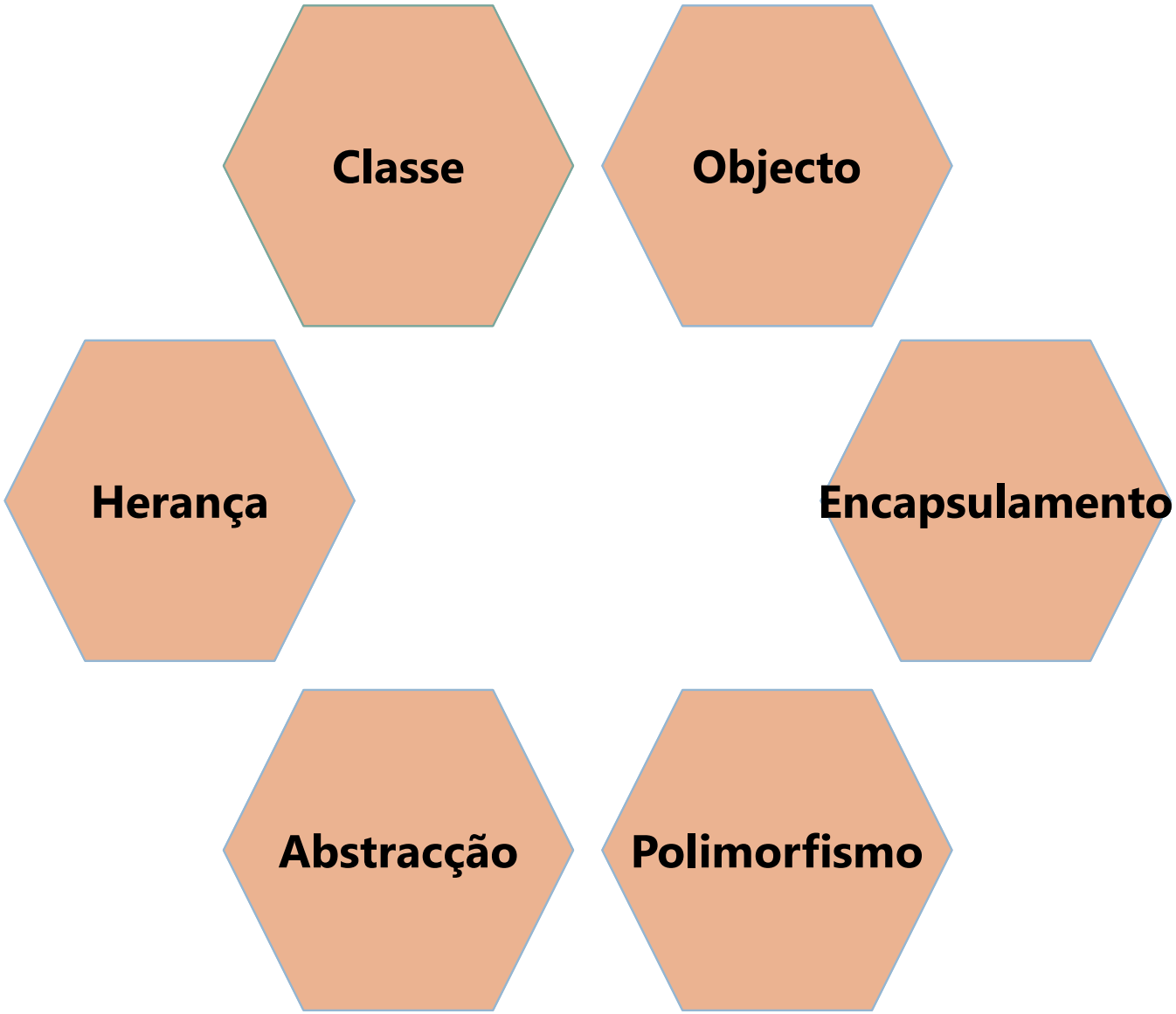

Programação Orientada Pelos Objectos
Princípios de Desenho Orientado pelos Objectos

Programação orientada pelos objectos



3 Princípio da Substituição

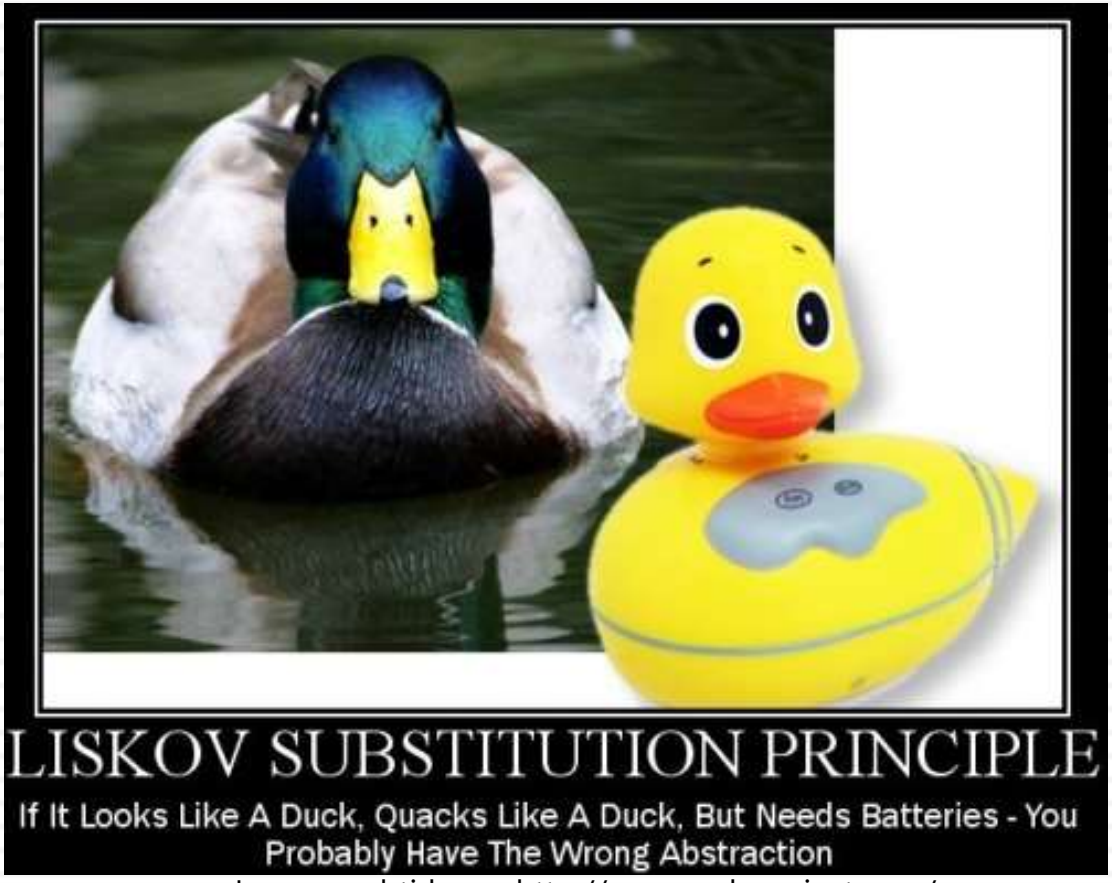


Imagem obtida em http://www.codeproject.com/

Princípio da Substituição

- *"Subtypes must be substitutable for their base types"*
- Requer que os sub-tipos se comportem de acordo com a especificação dos seus super-tipos

Princípio da Substituição

- *"Subtypes must be substitutable for their base types"*
- Requer que os sub-tipos se comportem de acordo com a especificação dos seus super-tipos

```
private static void printSpeech(Animal pet) {
   System.out.println(pet.speak());
}
```

```
Animal animal1 = new DogClass("Boby");
printSpeech(animal1);


Animal animal2 = new CatClass("Tareco");
printSpeech(animal2);
```

Princípio da Substituição


```
public class Counter {
  private int value;

  public Counter(int value) { this.value = value; }
  public int value() { return value; }
  public void inc(){ value++; }
  public void dec() { value--; }
}
```

Princípio da Substituição


```
public class Counter {
  private int value;

  public Counter(int value) { this.value = value; }
  public int value() { return value; }
  public void inc(){ value++; }
  public void dec() { value--; }
}
```

```
public class CounterMonitor extends Counter {
   private boolean status;

   public CounterMonitor(int x, boolean status) {
        super(x);
        this.status = status;
    }
   public void inc(){ if (status) super.inc(); }
   public void dec() { if (status) super.dec(); }
   public void turnOn() { status = true; }
   public void turnOf() { status = false; }
}
```

Princípio da Substituição

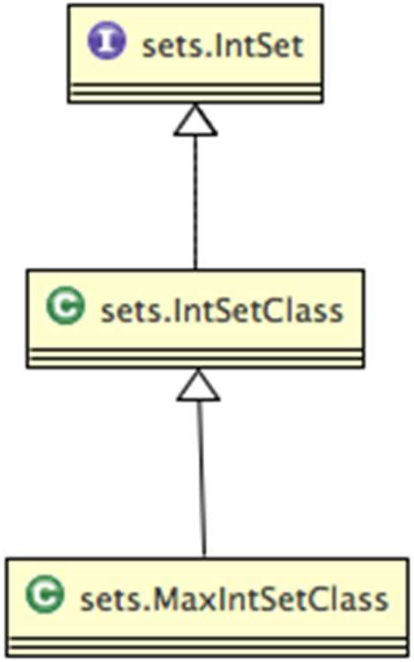

```
public class Main {

    public static void main(String[] args) {
        Counter a = new Counter(0);
        CounterMonitor b = new CounterMonitor(0,false);

        print(a);
        print(b);
    }

    private static void print(Counter counter) {
        System.out.println(counter.value());
    }
}
```

Princípio do Aberto/Fechado

Imagem obtida em http://www.codeproject.com/

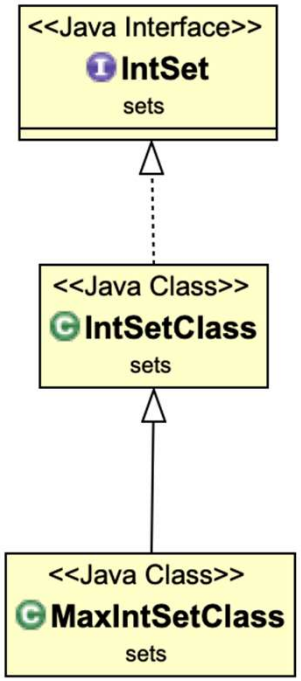
Princípio do Aberto/Fechado

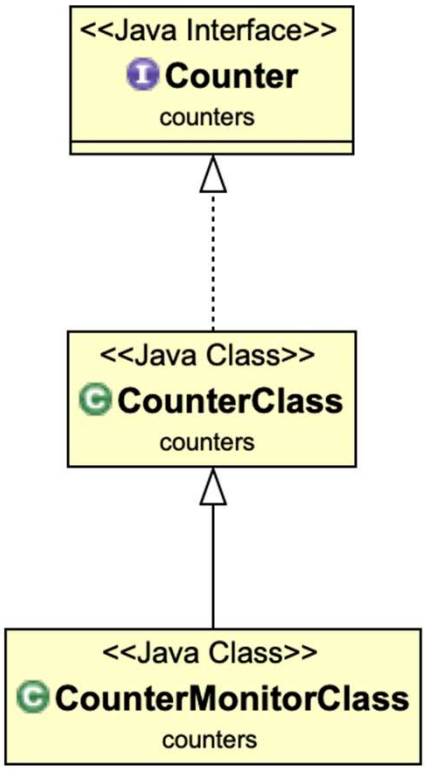
- *"Software entities should be open for extension, but closed for modification".*
- Adicionar novas funcionalidades ou alterações por extensão



Princípio do Aberto/Fechado

- *"Software entities should be open for extension, but closed for modification".*
- Adicionar novas funcionalidades ou alterações por extensão





12 Princípio da Segregação de Interfaces

Imagem obtida em http://www.codeproject.com/

Princípio da Segregação de Interfaces

- *Clients should not be forced to depend upon interfaces that they do not use.*
- Uma classe pode usada por diversos clientes, mas o protocolo de cada cliente pode ser diferente
  - Por exemplo, um cliente pode só usar um sub-conjunto dos métodos do protocolo

Princípio da Segregação de Interfaces


```
public interface FileAccess {
    boolean open(String name);
    boolean close();
    String read();
    void write(String text);
    void append(String text);
}
```

Princípio da Segregação de Interfaces


```
public interface FileAccess {
    boolean open(String name);
    boolean close();
    String read();
    void write(String text);
    void append(String text);
}
```

```
public class FileAccessClass implements FileAccess {
    private String name, content;
    private boolean open;

    public FileAccessClass() {...}
    public boolean open(String name) {...}
    public boolean close() {...}
    public String read() {...}
    public void write(String text) {...}
    public void append(String text) {...}
}
```

Princípio da Segregação de Interfaces


```
public static void main(String[] args) {
   FileAccess file = new FileAccessClass();

   if (file.open("text.txt")) {
        file.write(”vou apagar tudo...");
        file.close();
   }
}
```

Princípio da Segregação de Interfaces


```
public static void main(String[] args) {
   FileAccess file = new FileAccessClass();

   if (file.open("text.txt")) {
        file.write(”vou apagar tudo...");
        file.close();
   }
}
```

- E se esta entidade tiver que ser usada num contexto em que não deve ser possível escrever num ficheiro (e apagar o conteúdo)?

Princípio da Segregação de Interfaces

- Criar uma nova interface com um novo protocolo:

```
public interface FileLimitedAccess {
   boolean open(String file);
   boolean close();
   String read();
   void append(String text);
}
```

Princípio da Segregação de Interfaces

- Criar uma nova interface com um novo protocolo:

```
public interface FileLimitedAccess {
   boolean open(String file);
   boolean close();
   String read();
   void append(String text);
}
```

- A classe passa a implementar duas interfaces

```
public class FileAccessClass
       implements FileAccess, FileLimitedAccess {
    ...
}
```

Princípio da Segregação de Interfaces

- Objectos do tipo da interface mais restrita deixam de ter acesso ao método write

```
public static void main(String[] args) {
    FileAccess file = new FileAccessClass();
    FileLimitedAccess fileRestricted = new FileAccessClass();

    if (fileRestricted.open("text.txt")) {
        fileRestricted.write("vou apagar tudo...");
        fileRestricted.close();
    }
```

Princípio da Segregação de Interfaces

- Objectos do tipo da interface mais restrita deixam de ter acesso ao método write

```
public static void main(String[] args) {
    FileAccess file = new FileAccessClass();
    FileLimitedAccess fileRestricted = new FileAccessClass();

    if (fileRestricted.open("text.txt")) {
        fileRestricted.write("vou apagar tudo...");
        fileRestricted.close();
    }
```

The method write(String) is undefined for the type FileLimitedAccess

Princípio da Segregação de Interfaces

- Outro exemplo...
- Conjunto monotónico de inteiros

```
public interface IntSet {
   public void insert(int x);
   public void remove(int x);
   public boolean isIn(int x);
   public boolean subset(IntSet s);
   public int size();
   public Iterator elements();
}
```

Princípio da Segregação de Interfaces

- Outro exemplo...
- Conjunto monotónico de inteiros

```
public interface IntSet {
   public void insert(int x);
   public void remove(int x);
   public boolean isIn(int x);
   public boolean subset(IntSet s);
   public int size();
   public Iterator elements();
}
```

Princípio da Segregação de Interfaces

- Outro exemplo...
- Conjunto monotónico de inteiros

```
public interface IntSet {
   public void insert(int x);
   public void remove(int x);
   public boolean isIn(int x);
   public boolean subset(IntSet s);
   public int size();
   public Iterator elements();
}
```

```
public interface IntMonotonicSet {
   public void insert(int x);
   public boolean isIn(int x);
   public boolean subset(IntSet s);
   public int size();
   public Iterator elements();
}
```