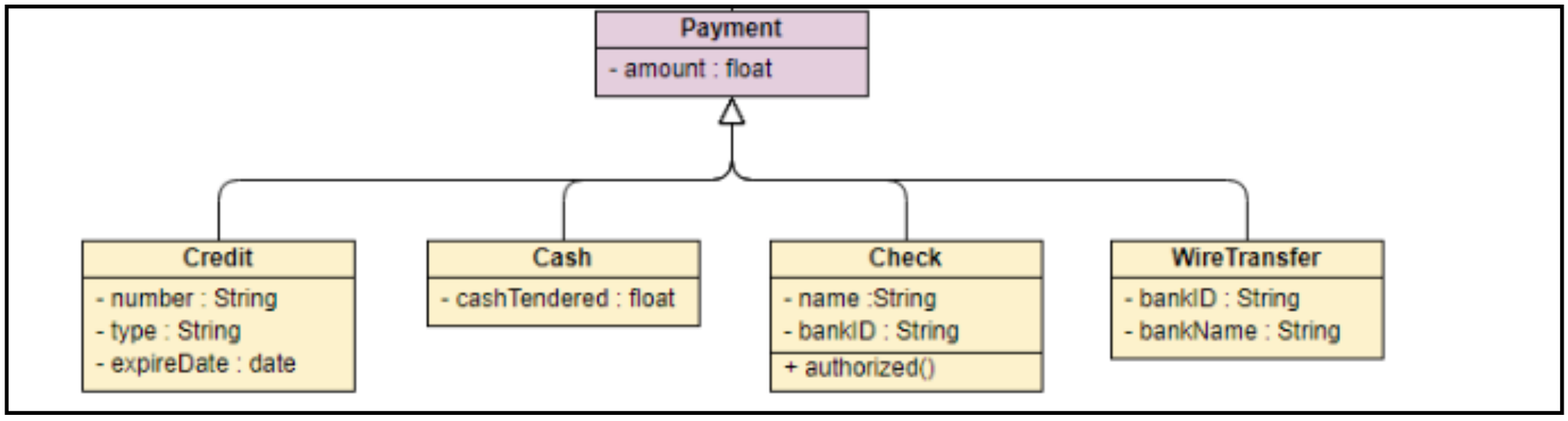
Relembrando…

1. Implemente a hierarquia de classes abaixo.



Relembrando…

1.

```
   public abstract class Payment {
13      protected float amount;
14
15      public float getAmount() {
16          return amount;
17      }
18
19      public void setAmount(float amount) {
20          this.amount = amount;
21      }
22
23      public Payment(float amount) {
24          this.amount = amount;
25      }
```

```
12  public class Cash extends Payment {
13      private float cashTendered;
14
15      public Cash(float cashTendered, float amount) {
16          super(amount);
17          this.cashTendered = cashTendered;
18      }
19
20      public float getCashTendered() {
21          return cashTendered;
22      }
23
24      public void setCashTendered(float cashTendered) {
25          this.cashTendered = cashTendered;
26      }
```

Relembrando…

1.

```
14    public class Check extends Payment {
15
16        private String name, bankID;
17        private boolean authorized;
18
19        public String getName() {
20            return name;
21        }
22
23        public void setName(String name) {
24            this.name = name;
25        }
26
27        public String getBankID() {
28            return bankID;
29        }
30
31        public void setBankID(String bankID) {
32            this.bankID = bankID;
33        }
34
35        public boolean isAuthorized() {
36            return authorized;
37        }
38
39        public void setAuthorized(boolean authorized) {
40            this.authorized = authorized;
41        }
```

Relembrando…

2. Crie um programa principal no qual o usuário possa escolher a forma de pagamento e informar os dados correspondentes.

```
14      public static void main(String[] args) throws IOException {
15          Payment p = null;
16          BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
17
18          int op;
19          do {
20              System.out.println("1 - Cash | 2 - Check | 3 - Credit | 4 - Wire Transfer | 5 - End Prog
21              op = Integer.parseInt(br.readLine());
22
23              switch (op) {
24                  case 1:
25                      p = new Cash(5f, 10f);
26                      break;
27                  case 2:
28                      p = new Check("BB", "1233", true, 100f);
29                      break;
30                  case 3:
31                      p = new Credit("1233231231232", "Credit", new Date(2019, 04, 11), 2000f);
32                      break;
33                  case 4:
34                      p = new WireTransfer("1233231231232", "BB", 1567.89f);
35                      break;
36                  case 5:
37                      System.out.println("Good bye...");
38                      break;
39                  default:
40                      System.err.println("Invalid option.");
41              }
42              if (op >= 1 && op <= 4) {
43                  System.out.println(p);
44              }
45          } while (op != 5);
46      }
```

Interface

Programação Orientada a Objetos

Rafael Vieira Coelho

O que seria uma interface?

- Não é uma classe.

- Só pode ter métodos abstratos (sem implementação).

- Nunca teremos um objeto pois não é uma classe.

Exemplo de Interface

- Podemos criar um arquivo com a nossa interface MyInterface (uma variável e um método abstrato):

```
public interface MyInterface {

    public String hello = "Hello";

    public void sayHello();
}
```

- E podemos acessar as variáveis definidas

```
System.out.println(MyInterface.hello);
```

Para implementar a interface

- Podemos criar um arquivo com a nossa interface MyInterface (uma variável e um método abstrato):

```
public class MyInterfaceImpl implements MyInterface {

    public void sayHello() {
        System.out.println(MyInterface.hello);
    }
}
```

- E podemos instanciar objetos de classes que implementam a interface:

```
MyInterface myInterface = new MyInterfaceImpl();

myInterface.sayHello();
```

Podemos implementar várias interfaces em uma mesma classe

```
public interface MyInterface {

    public String hello = "Hello";

    public void sayHello();
}
```

```
public interface MyOtherInterface {

    public void sayGoodbye();
}
```

```
public class MyInterfaceImpl
    implements MyInterface, MyOtherInterface {

    public void sayHello() {
        System.out.println("Hello");
    }

    public void sayGoodbye() {
        System.out.println("Goodbye");
    }
}
```

Métodos default de interfaces

- A partir de Java 8, é possível criar métodos padrão (default) implementados em interfaces.

```
public interface Operations {

    int add(int a, int b);

    default int add(int a, int b) {
        return a + b;
    }

}
```

Métodos Estáticos de interfaces

- A partir de Java 8, também é possível criar métodos estáticos implementados em interfaces.

```
public interface MyInterface {

    public static void print(String text){
        System.out.print(text);
    }
}
```

Herança entre interfaces

```
public interface MySuperInterface {

    public void sayHello();

}
```

```
public interface MySubInterface extends MySuperInterface {

    public void sayGoodbye();
}
```

```
public class MyClass implements MySubInterface {

    public void sayHello() {
        System.out.println(“Hi”);
    }

    public void sayGoodbye() {
        System.out.println(“bye”);
    }
}
```

Generics

- Podemos definir na declaração da interface que ela receberá o tipo de objeto genérico que será usado.

```
public interface MyProducer <T>{

    public T produce();

}
```

```
public class Car {
    ...
}
```

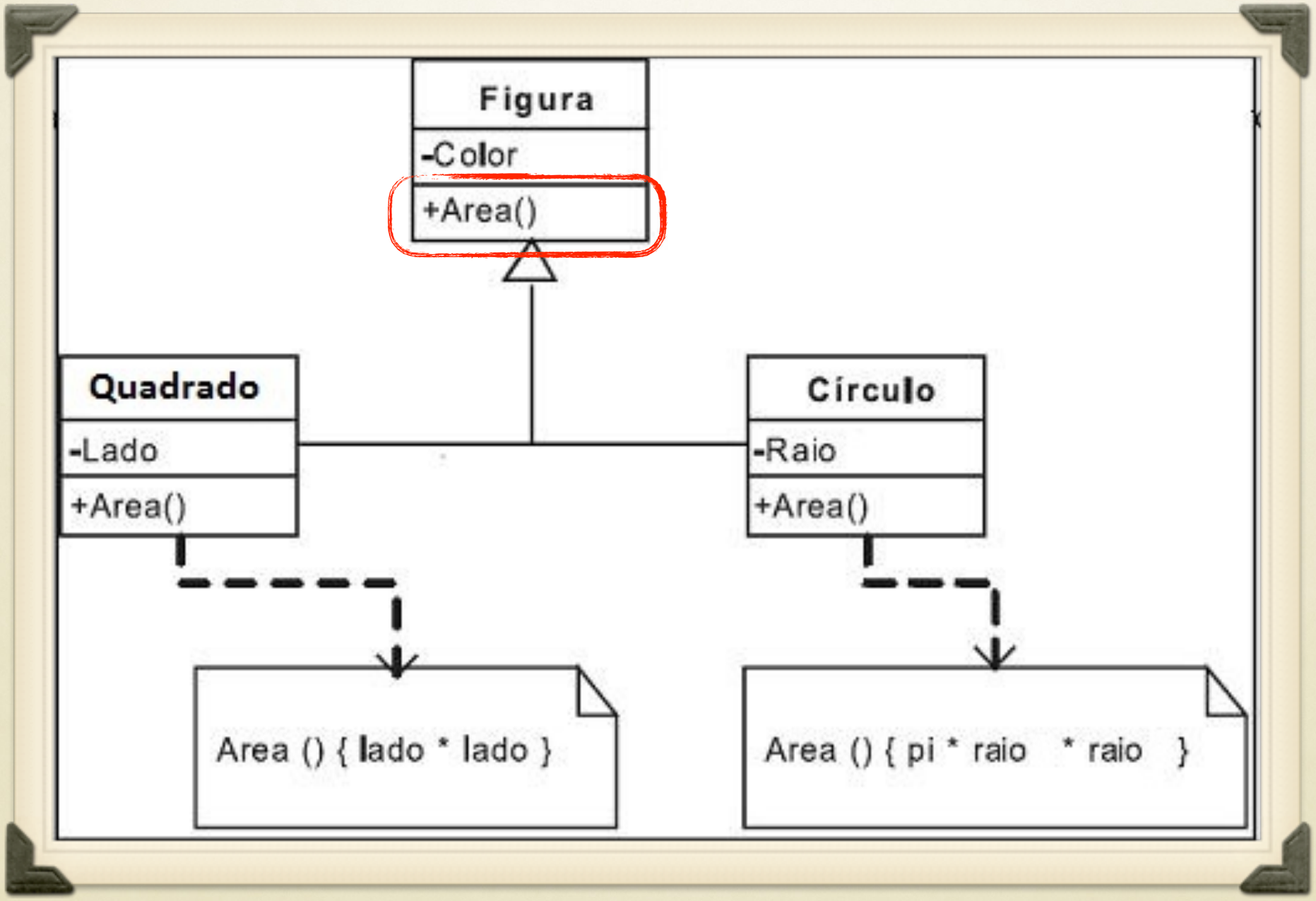
```
public class CarProducer<T> implements MyProducer<T>{

    @Override
    public T produce() {
        return (T) new Car();
    }
}
```

```
public class CarTest {
    public static void main(String []args) {
        MyProducer<Car> myCarProducer = new CarProducer<Car>();
        Car produce = myCarProducer.produce();
    }
}
```

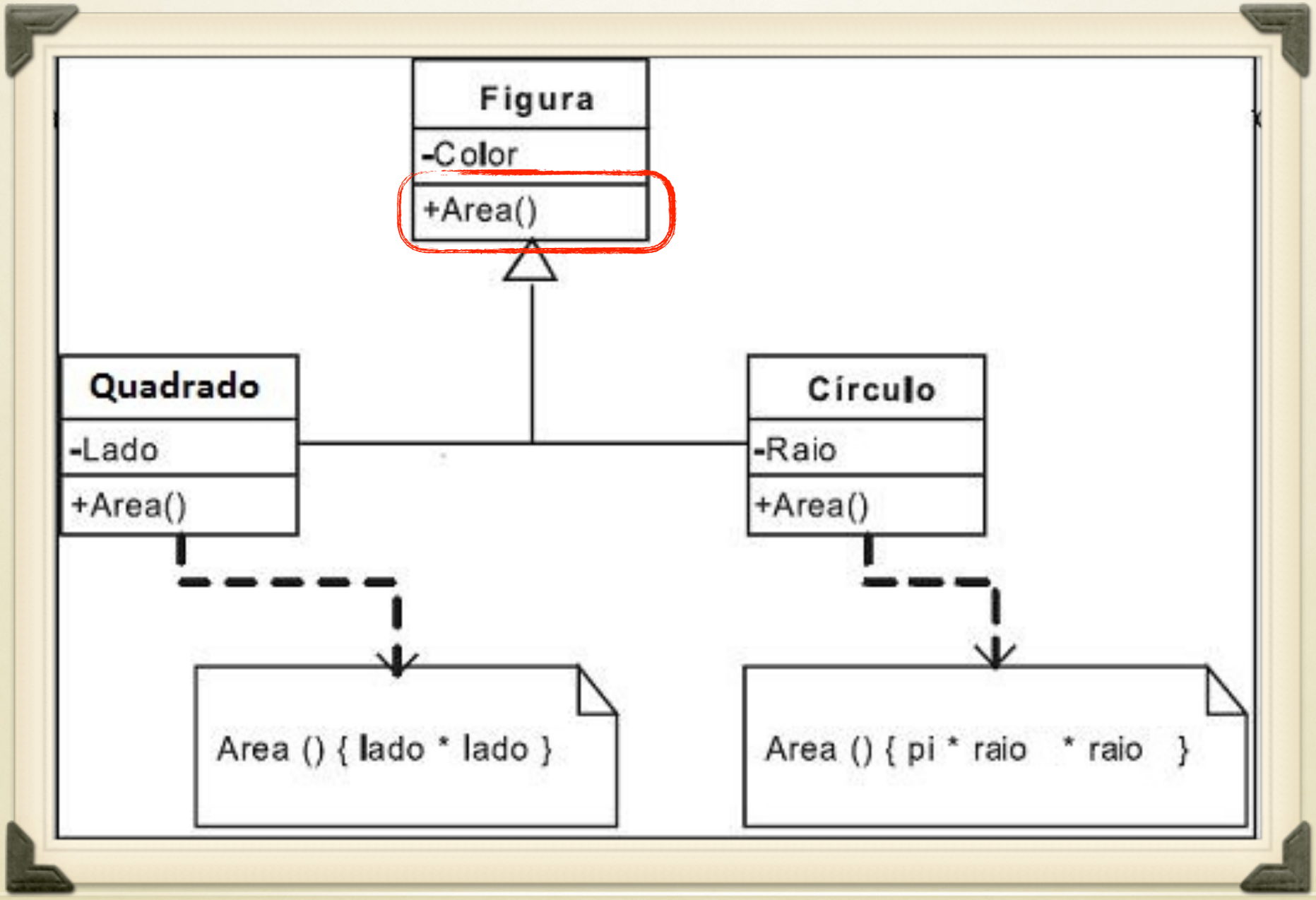
Implementaremos

- Tanto Quadrado quanto Circulo tem o mesmo método Area().



Duas possibilidades

- Podemos colocar o método abstrato na classe abstrata Figura.



Duas possibilidades

- Ou fazer com que as classes filhas implementem uma Interface com o método Area().

```
interface IArea {

    double Area();
}
```

```
public class Circulo implements IArea {

    protected double radius;
    protected String color;

    public Circulo() {
        radius = 1.0;
        color = "red";
    }

    public Circulo(double r) {
        radius = r;
        color = "red";
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }
@Override
public double Area() {
    return Math.pow(this.radius, 2) * Math.PI;
}
```

Duas possibilidades

- Ou fazer com que as classes filhas implementem uma Interface com o método Area().

```
interface IArea {

    double Area();
}
```

```
12    public class Quadrado implements IArea {
13        private double lado;
14
15        @Override
          public double Area() {
17            return lado * lado;
18        }
19
20        public double getLado() {
21            return lado;
22        }
23
24        public void setLado(double lado) {
25            this.lado = lado;
26        }
27
28
29    }
```

Quando usar Interface e quando usar Classes Abstratas?

- **Classes Abstratas:**
  - Você precisa compartilhar código entre mais de uma classe que tem relação (herança)
  - A sua classe mãe da herança nunca irá ser instanciada.
- **Interfaces**:
  - Você já usou herança (extends) e precisa fazer um comportamento semelhante entre várias classes.
  - Você precisa compartilhar um comportamento entre classes que não tem relação (não existe herança).

Java tem algumas interfaces populares

- **Comparable:** deve implementar um método que permite comparar com outro objeto.

- **Iterable:** usado para percorrer uma coleção de objetos através de um iterador.

- **List:** representa uma lista e exige a implementação de vários métodos (add, remove, clear, etc.). Exemplos de implementações: ArrayList e LinkedList.

http://blog.amitinside.com/Java-_able-Interfaces/

Tarefas

1. Complemente a hierarquia de classes passada anteriormente, criando uma classe Cilindro (atributo altura) que deve ser filha de Circulo. O volume do cilindro é calculado por:

$$Volume = \pi \times raio^2 \times altura$$

Já a área do Circulo é dada por:

$$Area = \pi \times raio^2$$

2. Crie um menu de opções no qual o usuário pode cadastrar 1000 cilindros. Para isto, utilize um vetor de objetos Cilindro. O usuário pode escolher calcular o volume do cilindro a partir da posição desejada no vetor. Além disso, ele pode remover cilindros a partir do valor de seu raio (ou seja, serão removidos todos os cilindros com o raio informado pelo usuário). Por fim, o usuário pode escolher sair do programa. Modularize o seu código através de métodos estáticos.