Padrões de Projeto

Unidade 1

Edison Silva e
Valter Camargo

INFORMAÇÃO
TECNOLOGIA
& INOVAÇÃO

INTRODUCTION

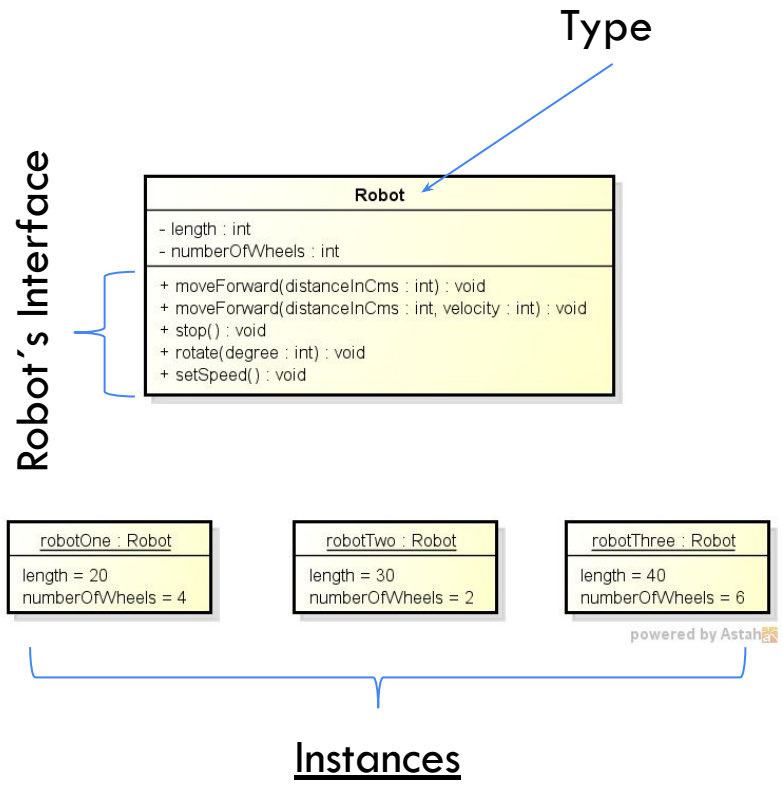
- What is a design pattern ?
- How to decompose a system into objects ?
  - Nouns ?
  - Collaborations among objects ?
  - Responsibilities ?
  - Real world ?
- Modelling a system as the real world may represent the present, but not the future ☹
- The **abstractions** used during the project are the key for flexible designs

INTRODUCTION

- Design Patterns assist in finding out not so obvious abstractions
  - States,
  - Strategies,
  - Observers,
  - Composites,
  - Etc..
- Would you create a class with these names in your project ?

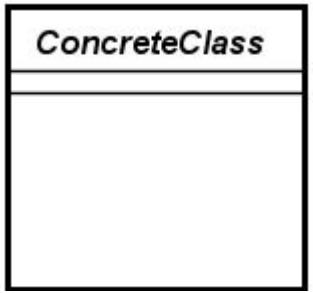
INTERFACES

- Interfaces
  - Interfaces are the set of signatures an object exposes
  - The interface of an object states all the requisitions can be sent to it
  - A Type is a name used to denote a specific interface
  - Objects can have several types (hierarchy)
  - Interfaces do not say anything about the object's implementation (i.e., the o methods)



STATIC OR DYNAMIC BINDING ? POLYMORPHISM ?

- Static Binding / Dynamic
  - Static: The link between a **request** to an object **to one of its operations** is in design/compile time
  - Dynamic: The link between a **request** to an object **to one of its operations** is in run-time




```
Client c = new Client
...
c.myMethod()
```

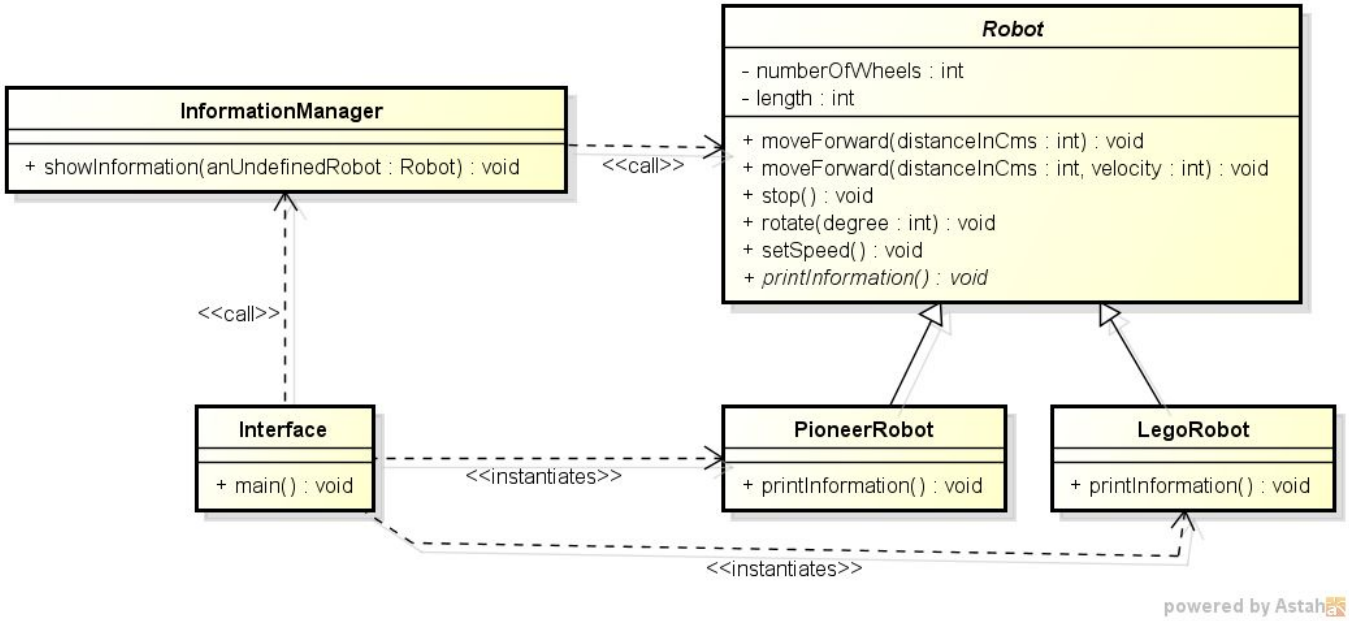
Client não é abstrata/interface
Os serviços oferecidos são "linkados" ao tipo concreto antecipadamente… em tempo de compilação


```
Abstract Client c;
...
c.myMethod()
```

Client é abstrata
Os serviços oferecidos podem ser demandados mesmo sem saber quem
irá efetivamente executá-los.

STATIC OR DYNAMIC BINDING ? POLYMORPHISM ?

- Let´s understand this example :



STATIC OR DYNAMIC BINDING ? POLYMORPHISM ?

- What is “Dynamic binding” ?
  - The link between a **request** to an object **to one of its operations** in runtime (thanks to polymorphism)

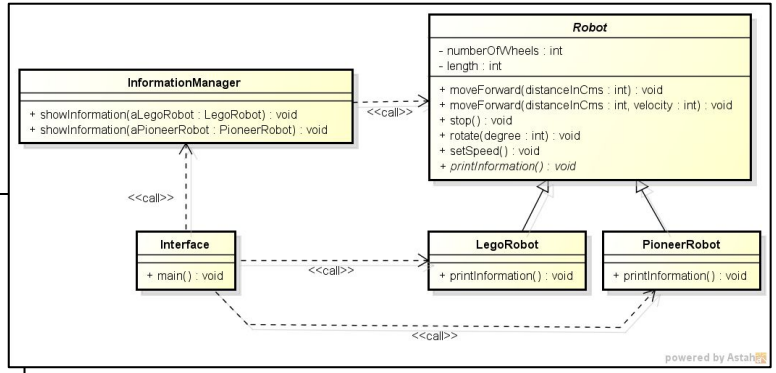
```
public static void main(String args[]){

    Robot robotOne;

    if (condition) {
        robotOne = new LegoRobot();
    } else {
        robotOne = new PioneerRobot();
    }

    InformationManager.showInformation(robotOne);
}
```

Important !

Aqui está a mágica

```
public class PioneerRobot extends Robot {

@Override
public void printInformation() {
      System.out.println(
            "This is a pionner robot"
       );
}
```

STATIC BINDING...




```java
public class InformationManager {

    public static void showInformation(LegoRobot aLegoRobot){
        aLegoRobot.printInformation();
    }

    public static void showInformation(PioneerRobot aPioneerRobot){
        aPioneerRobot.printInformation();
     }
}
```

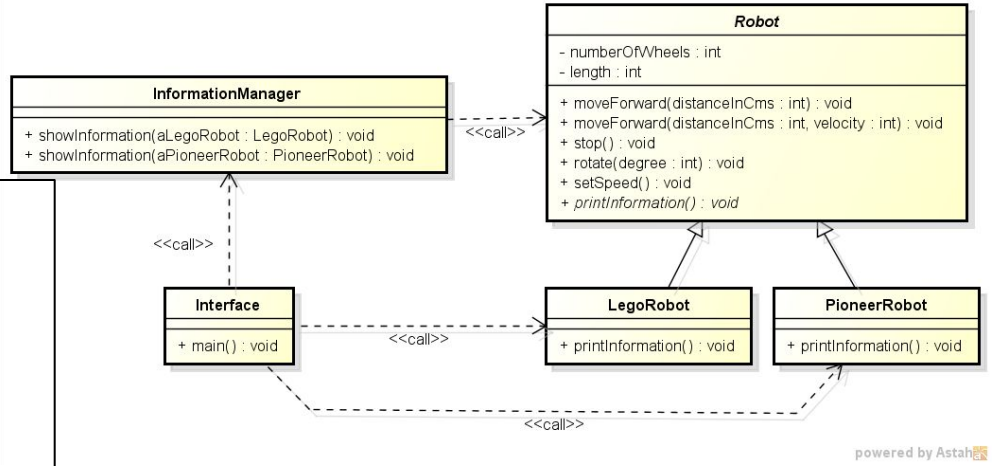
```java
public class PioneerRobot extends Robot  {
   public void printInformation() {
         System.out.println("This is a pionner robot");
    }
}
```

```java
public class LegoRobot extends Robot {
    public void printInformation() {
         System.out.println("This is a lego robot !");
    }
}
```

```java
public static void main(String args[]){

PioneerRobot pioneerRobot = new PioneerRobot();
LegoRobot aLegoRobot = new LegoRobot();

String type = "null";

if (type == "pioneer"){
   ShowInformation.showInformation(pioneerRobot);
} else {
   ShowInformation.showInformation(aLegoRobot);
}
}
```

STATIC BINDING...

```
class InformationManager:
    def showInformation(aLegoRobot: LegoRobot):
        aLegoRobot.printInformation()

    def showInformation(aPioneerRobot: PioneerRobot):
        aPioneerRobot.printInformation()
}
```




```
class Robot:
    def printInformation(__self__):
        raise NotImplementedError

class PioneerRobot(Robot):
    def printInformation(__self__):
        print("This is a pionner robot")

class LegoRobot(Robot):
    def printInformation(__self__):
        print("This is a lego robot")
```

```
if __name__ == "__main__":
    if condition:
        robotOne = LegoRobot()
    else:
        robotOne = PioneerRobot()

    InformationManager.showInformation(robotOne)
```

YOU NEED TO PROGRAM FOR INTERFACES...

- **You need to program for interfaces, not for particular implementations**
  - Using inheritance you can define *family of objects* sharing identical interfaces
  - All subclasses are able to reply to the requests forwarded to the interface of the abstract class
  - The main benefit of manipulating objects only considering their interfaces is that clients keep **unaware** of the type of the objects they use

GAMMA´S DESIGN PATTERNS

		Purpose		
		Creational	Structural	Behavioral
Escopo	Class	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

TEMPLATE FOR DESCRIBING PATTERNS

- Pattern Name and Classification
- Intent
- Also Known As
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Sample Code
- Known Uses
- Related Patterns

Singleton

(Creational pattern)

INFORMAÇÃO
TECNOLOGIA
& INOVAÇÃO

SINGLETON

When to use ?

You should use this pattern when classes must have just one instance

SINGLETON

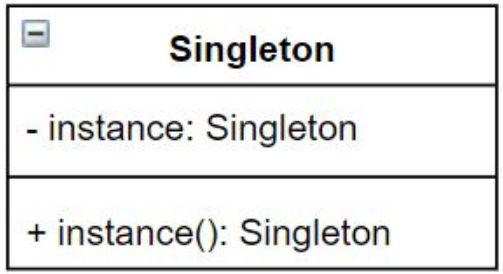
Examples

Classes responsible for establishing connections with DB. It is not necessary create a new connection object every time you need to connect to the DB… this will overload memory and make your code polluted. You can use the same object every time.

Classes that represent sensors in robots

Classes responsible for logging events of the system execution

SINGLETON

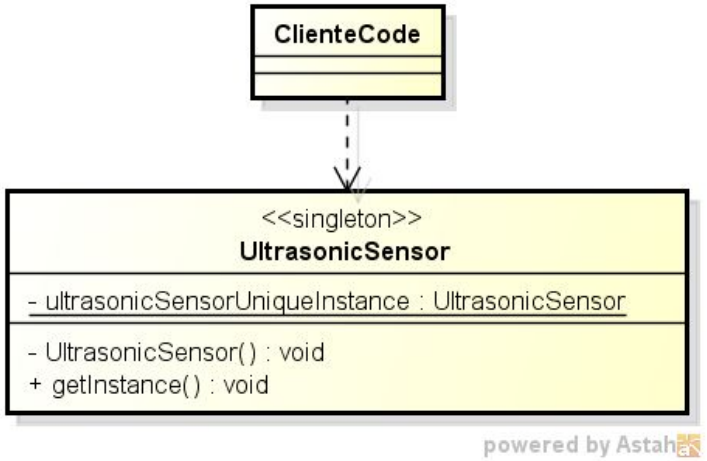



```
public class Singleton {

    private static Singleton instance;
    private Singleton() {
      this.instance = new Singleton()
    }

    public Singleton getInstance(){
      if (instance == null)
      {
            instance= new Singleton();
      }
      return instance;
}}
```

```
class Singleton:

    _instance = None

    @classmethod
    def instance(cls): #cls == class
        if cls. instance is None:
            cls. instance = cls()
        return cls._instance
```

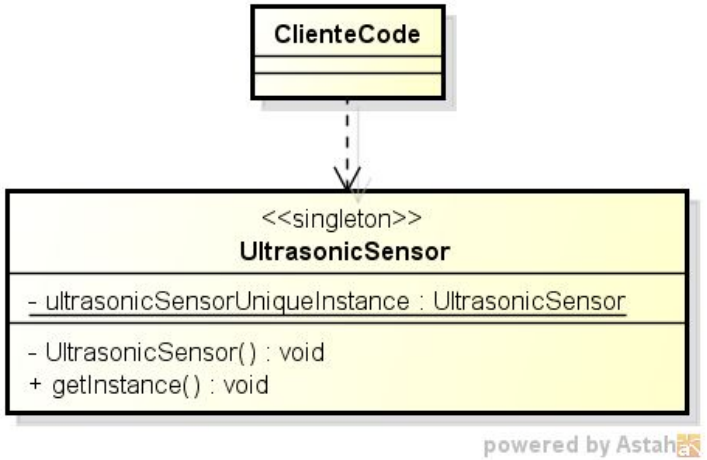
ROBOT SENSOR - SINGLETON IN JAVA

```
public class UltrasonicSensor {

    private static UltrasonicSensor ultrasonicSensorUniqueInstance;

    private UltrasonicSensor()  { ... }

    public static UltrasonicSensor getInstance(){

        if (ultrasonicSensorUniqueInstance ==null)
        {
                synchronized (UltrasonicSensor.class)
                {
                    if (ultrasonicSensorUniqueInstance ==null) {
                        ultrasonicSensorUniqueInstance =new UltrasonicSensor();
            }       }
        return ultrasonicSensorUniqueInstance;
    }
}
```



ROBOT SENSOR - SINGLETON IN PYTHON

```
class UltrasonicSensor:

    _ultrasonicSensorinstance =None

    @classmethod
    def instance(cls): #cls == class
        if (cls.ultrasonicSensorinstance is None):
            cls.ultrasonicSensorinstance= cls
        return cls.ultrasonicSensorinstance

if __name__ == "__main__":
    ultrasonicSensor1 = UltrasonicSensor.instance()
    ultrasonicSensor2 = UltrasonicSensor.instance()

    print(ultrasonicSensor1 == ultrasonicSensor2)#true
```



DB CONNECTION - SINGLETON IN JAVA

```java
public class DatabaseConnection {

    private static DatabaseConnection instance = null;

    private DatabaseConnection() {
                    // Establish a database connection
            ...
        }

        public synchronized static DatabaseConnection instance() {
            if( instance == null ) {
                instance =new DatabaseConnection( );
            }
            return instance;

            public synchronized String getAssignmentGrade (String student, String course, String assignment) {
                ...
                return ...;
            }
        }
    }
}
```

```python
class Singleton:

    _instance = None

    @classmethod
    def instance(cls): #cls == class
        if cls._instance is None:
            cls._instance = cls()
        return cls._instance
```

DB CONNECTION - SINGLETON IN JAVA

```
public class Client {

    public void printTranscript(String student) {

      DatabaseConnection db = DatabaseConnection.instance();
      ...
      String grade = db.getAssignmentGrade (student, course, assignment);
      ...
    }
}
```

LET´S PRACTICE ?

You must reimplement the DB example (provided in Java) using the Python language

Strategy
(Behavioral pattern)

INFORMAÇÃO
TECNOLOGIA
& INOVAÇÃO

STRATEGY

When to use?

Use the Strategy pattern when you want to use different variants of an algorithm (family of algorithms) and be able to switch between them during execution.

STRATEGY

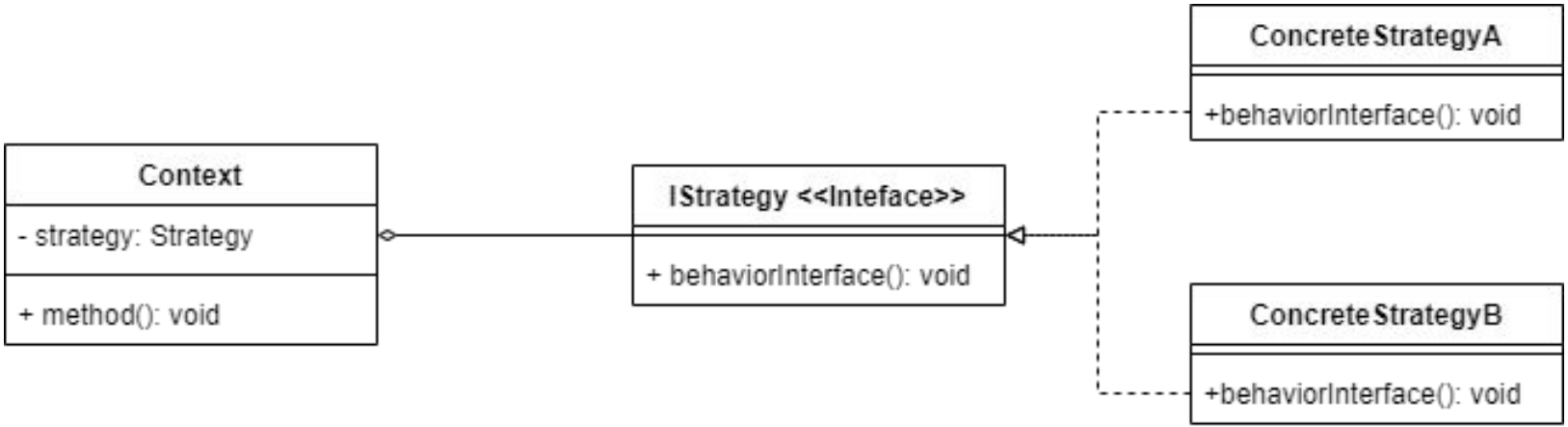
Example:

Consider the example of sorting. We implemented bubble sort, but the data started to grow and bubble sort started getting very slow.

In order to tackle this we implemented Quick sort. But now although the quick sort algorithm was doing better for large datasets, it was very slow for smaller datasets.

In order to handle this we implemented a strategy **where for small datasets, bubble sort will be used and for larger, quick sort**.

STRATEGY



STRATEGY IN PYTHON

```python
class Strategy:
    def behavior(self) -> None:
        raise NotImplementedError


class ConcreteStrategyA(Strategy):
    def behavior(self) -> None:
        Pass


class ConcreteStrategyB(Strategy):
    def behavior(self) -> None:
        Pass
```

```python
class Context():

    def __init__(self, strategy: Strategy) -> None:
        self.__strategy = strategy

    def method(self) -> None:
        self.__strategy.behavior()
```

STRATEGY IN JAVA

```
public class  Context {
      private strategy IStrategy;
      ...
}
```

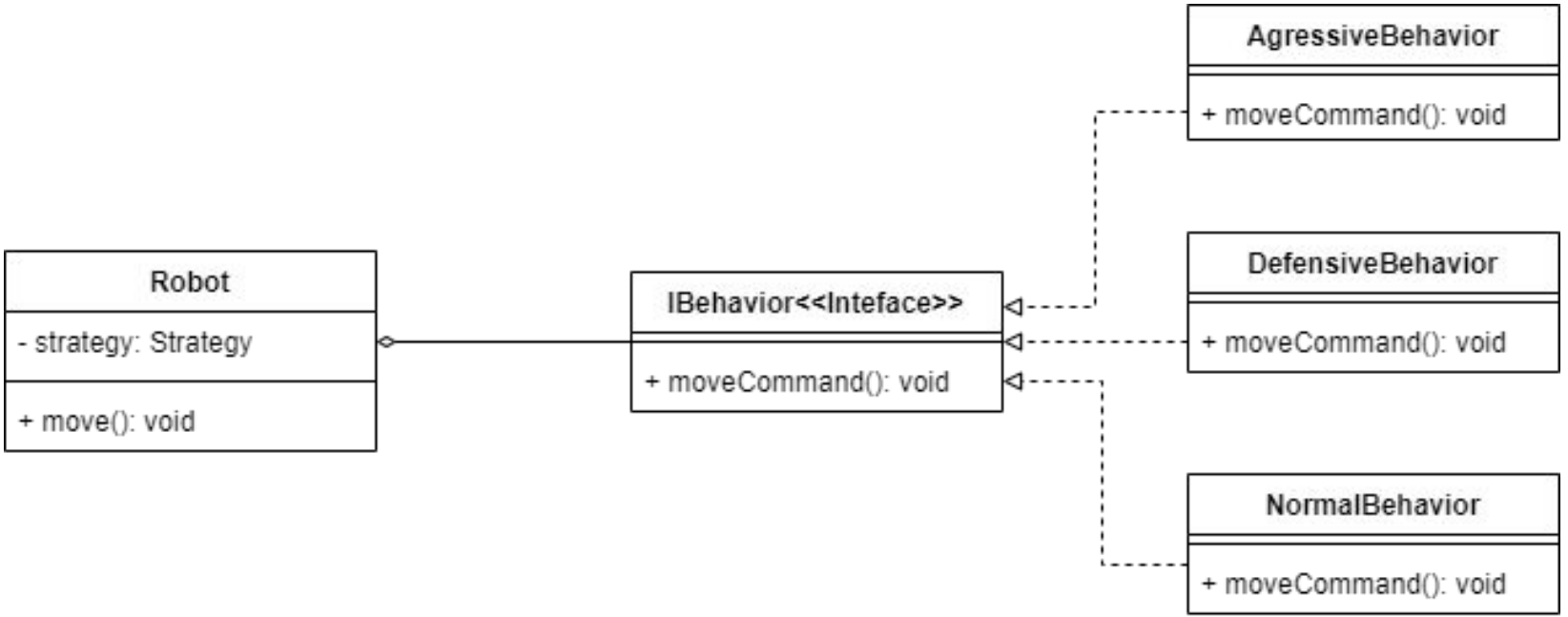
```
public abstract  class IStrategy {

      public abstract void behaviorInterface();
}
```

```
public class ConcreteStrategyA extends IStrategy {

     public void behaviorInterface(){
     //code for behaviorInterface A
     }
}
```

```
public class ConcreteStrategyB extends IStrategy {

     public void behaviorInterface(){
     //code for behaviorInterface B
     }
}
```

```
public class ConcreteStrategyC extends IStrategy {

     public void behaviorInterface(){
     //code for behaviorInterface C
     }
}
```

ROBOT EXAMPLE



ROBOT EXAMPLE - IN JAVA

```
public interface IBehaviour {
      public int moveCommand();
}
```

```
public class DefensiveBehaviour implements Ibehaviour {
   public int moveCommand()
   {
     System.out.println("\tDefensive Behaviour: if find
      another robot run from it");
     return -1;
    }
}
```

```
public class AgressiveBehaviour implements Ibe

     public int moveCommand()
    {
              System.out.println("\tAgressive B
another robot attack it");
             return 1;
     }
}
```

```
public class NormalBehaviour implements Ibehaviour {

    public int moveCommand()
    {
        System.out.println("\tNormal Behaviour: if find
        another robot ignore it");
        return 0;
     }
}
```

ROBOT EXAMPLE - IN JAVA

```
public class Robot {

    IBehaviour behaviour;
    String name;
    public Robot(String name)
    {
         ...
    }

    public void move()
    {
        ...
        int command = behaviour.moveCommand();
        ....
     }
.....
```

Represent the behavior of the class, i.e., all methods that would be here, they are not anymore

It is a way of separating the structure from the behavior. The class is less propense to receive modifications

ROBOT EXAMPLE - IN JAVA

```
public class Main {
public static void main(String[] args) {
     Robot r1 = new Robot("Big Robot");
     Robot r2 = new Robot("George v.2.1");
     Robot r3 = new Robot("R2");

     r1.setBehaviour(new AgressiveBehaviour());
     r2.setBehaviour(new DefensiveBehaviour());
     r3.setBehaviour(new NormalBehaviour());
     r1.move();
     r2.move();
     r3.move();
     if (some condition)
          r1.setBehaviour(new DefensiveBehaviour());
     ....
     if (some condition)
          r2.setBehaviour(new AgressiveBehaviour());


     r1.move();
     r2.move();
     r3.move();
   }
}
```

Initial behavior (Default)

Behavioral changes in runtime

Behavioral changes in runtime

ROBOT EXAMPLE - IN PYTHON

```
class BehaviorStrategy:

    def moveCommand(self) -> None:
        raise NotImplementedError
```

```
class DefensiveBehavior(BehaviorStrategy):
    def moveCommand(self) -> None:
        print("Defensive Behaviour: if find another robot run from it)"
```

```
class AgressiveBehavior(BehaviorStrategy):
    def moveCommand(self) -> None:
        print("Agressive Behaviour: if find
```

```
class NormalBehavior(BehaviorStrategy):
    def moveCommand(self) -> None:
        print("Normal Behaviour: if find another robot ignore it)"
```

ROBOT EXAMPLE - IN PYTHON

```
if __name__ == "__main__":
    robot1 = Robot("Big Robot")
    robot2 = Robot("George v2.1")
    robot3 = Robot("R2")

    robot1.behavior(AgressiveBehavior())
    robot2.behavior(DefensiveBehavior())
    robot3.behavior(NormalBehavior())

    robot1.move()
    robot2.move()
    robot3.move()

    if condition:
        robot1.behavior(DefensiveBehavior())
    ...
    if condition:
        robot2.behavior(AgressiveBehavior())

    robot1.move()
    robot2.move()
    robot3.move()
```

Initial behavior (Default)

Behavioral changes in runtime

Behavioral changes in runtime

LET´S PRACTICE ?

Suppose a system that generates “safe routes” for pedestrians. However, the system must adapt itself in runtime as the person walk for streets. As the intention is to generate safe routes, the context can change in runtime, for example, if an attack or robbery happened in the generated route. When the context change, the system must be smart enough for proposing a new route.

There are different ways (algorithms) for generating routes:
1 - generates a not-to-save but a shortest route
2 - generate a safe route in an acceptable distance
3 - generate a super-safe-route, but usually the distance is the largest
Depending on the context, the generation of the route can change in runtime.