

SOLID

**Uma introdução ao
conjunto de princípios
fundamentais para**

**O desenvolvimento
de software saudável**

Princípios sólidos

Princípios sólidos

Organização

Objetivos do

SOLID

- Tolerar mudanças
- Sejam faceis de entender
- Sejam a base de código que possa ser usada em muitos sistemas de software

Breve

SOLID

história



Robert C Martin (signatário do Manifesto Agil, autor do da série de livros código limpo e desenvolvedor desde a década de 70), vem escrevendo sobre qualidade de software a muito tempo e em 2000 já havia estabelecido um conjunto de princípios e práticas em seus trabalhos e publicações.

Foi aí que em 2004, Michael Feathers (um importante e antigo desenvolvedor da comunidade C++) percebeu que se reorganizasse os princípios, as primeiras letras de cada princípio poderiam formar a palavra SOLID.

Assim nasceu os princípios SOLID.

S O L I D

S **O** **L** **I** **D**

Single

Responsability

Principle

SRP

Principio

Responsabilidade

Unica

SRP

```
public class Robot {  
    void cook() { ... }  
    void text() { ... }  
    void state() { ... }  
    void image() { ... }  
    void video() { ... }  
    void youtube() { ... }  
}
```

SRP

```
public class RobotChef {  
    void cook() { ... }  
}  
  
public class RobotGardener {  
    void clean() { ... }  
}  
  
public class RobotPainter {  
    void paint() { ... }  
}  
  
public class RobotDriver {  
    void drive() { ... }  
}
```

SOLID

Open-Closed

Principle

OCP

Princípio

Aberto-Fechado

OCP

```
public class Robot {  
    void exec() {  
        this.skill.cut();  
    }  
}
```

// 

```
public class Robot {  
    void exec() {  
        this.skill.paint();  
    }  
}
```

OCP

```
public class Robot {  
    Robot(Array<Skill> skills){  
        this.skills = skills;  
    }  
    void exec() {  
        for (Skill skill: this.skills){  
            skill.exec();  
        }  
    }  
}
```

OCP

```
public class CookSkill
    implements Skill {

    void exec() { ... }
}

public class PaintSkill
    implements Skill {

    void exec() { ... }
}
// ... more skills

// using
Robot robot = new Robot([
    new CookSkill(),
    new PaintSkill()
]);
```


SOLID

Liskov

Substitution

Principle

LSP

Princípio

Substituição

Liskov

LSP

LSP

```
public class Sam {  
    Coffee cook() { ... }  
}  
  
public class Eden extends Sam {  
    Water cook() { ... }  
}
```

LSP

```
public class Sam {  
    Coffee cook() { ... }  
}  
  
public class Eden extends Sam {  
    Coffee cook() {  
        // cappuccino is a Coffee type  
        return cappuccino;  
    }  
}
```

S O L 1 D

Interface

Segregation

Principle

ISP

Princípio

Segregação

Interface

ISP

```
public interface Robot {  
    void sping();  
    void rotateArms();  
    void wiggleAntenas();  
}  
  
public class Eden implements Robot {  
    Pair<Arm, Arm> arms;  
    Pair<Antena, Antena> antenas;  
  
    void sping() { ... }  
    void rotateArms() { ... }  
    void wiggleAntenas() { ... }  
}
```


ISP

```
public class Sam
    implements Robot {

    Pair<Arm, Arm> arms;

    void sping() { ... }
    void rotateArms() { ... }

    // NÃO TENHO ANTENAS
    void wiggleAntenas() {
        throw new
            Exception("Não tenho antenas!");
    }
}
```

ISP

```
public interface SpinRobot {  
    void spin();  
}  
  
public interface ArmsRobot {  
    Pair<Arm, Arm> arms;  
    void rotate();  
}  
  
public interface AntenasRobot {  
    Pair<Antena, Antena> antenas;  
    void wiggleAntenas();  
}
```

ISP

```
public class Eden implements
    SpinRobot,
    ArmsRobot,
    AntenasRobot {

    Pair<Arm, Arm> arms;
    Pair<Antena, Antena> antenas;

    void sping() { ... }
    void rotateArms() { ... }
    void wiggleAntenas() { ... }

}

public class Sam implements
    SpinRobot,
    ArmsRobot {

    Pair<Arm, Arm> arms;

    void sping() { ... }
    void rotateArms() { ... }

}
```

S O L I *D*

Dependency

Inversion

Principle

DIP

Princípio

Inversão

Dependência

