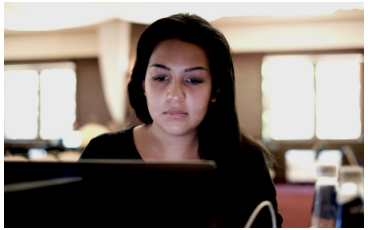
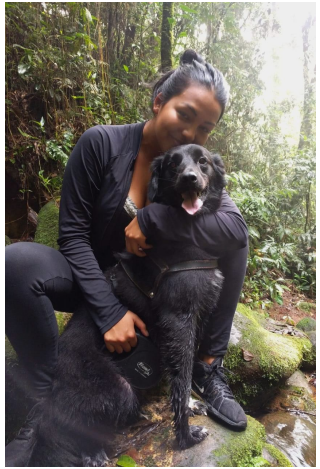
Refactoring e Redesign: Quando e como reescrever código?

Milena Mayumi





Milena Mayumi

Software Engineer @ SumUp

linkedin.com/in/milena-mayumi-costa

milena.mayumi@sumup.com

milenamayumi

O quê?

Refactor x Redesign

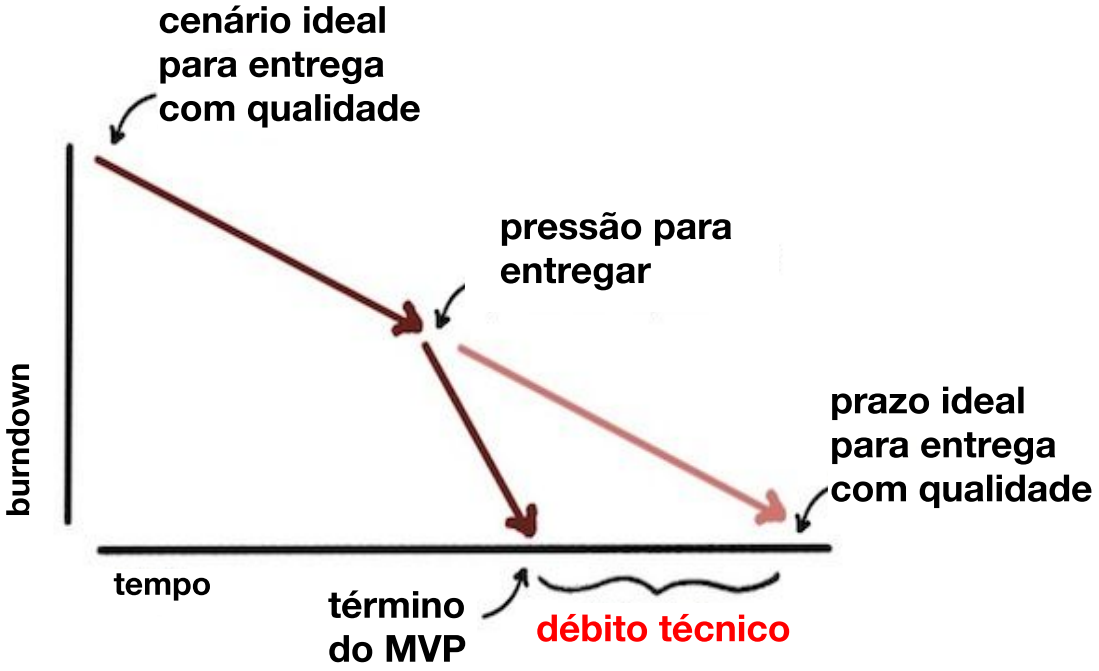
Quando?

Como?

OBJETIVO **Identificar e executar refactoring e redesign**

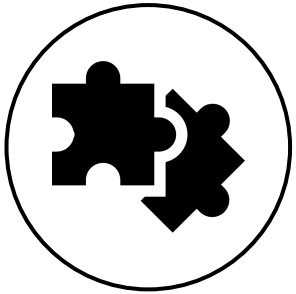
Contexto:

Melhoria de código sempre será necessária



Refactor

x



Redesign

"**Refactoring** é o processo de modificação de software tal que não altera o comportamento externo ainda que melhore a estrutura interna.*"

Martin Fowler

* Refactoring: Improving the Design of Existing Code. (2002)

Exemplos
Refactor

Adicionar logs

Adicionar testes

Corrigir um bug

Melhorar performance

Exemplos
Refactor

Deletar código

Reutilizar código

Renomear variáveis

Extrair um método

"[...] No contexto de melhoria de software, revisite o problema, o especifique novamente e, então, faça o **redesign** da solução para alinhar com o estado atual do negócio.*"

Gary Stonerock

* Redesigning vs. Refactoring Software Solutions.

Exemplos
Redesign

Renomear coisas

Mudar código de lugar

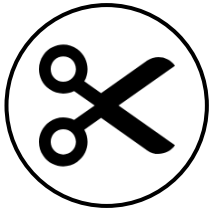
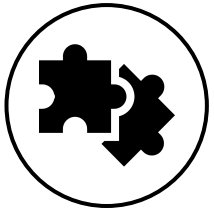
Reorganizar diretórios

Exemplos
Redesign

Extender funcionalidade

Mudar comunicação

Adicionar constraints

	 Refactor	 Redesign
MOTIVAÇÃO	Compreensão	Novo requisito
COMPLEXIDADE	Mudanças atômicas	Mudanças complexas
ESCOPO	Trecho de código	Fluxo
TEMPO	Horas	Dias
CAUSA	Débito técnico	MVP
RECURSOS	Keyboard	Whiteboard

O quê?

Quando?

Como?

OBJETIVO **Identificar e executar refactoring e redesign**

Quando fazer um refactoring?

Sempre!*

Quando fazer um refactoring?

Sempre!*

*Prioridade:
Código com frequência alta de mudanças

Quando fazer um redesign?

Por

necessidade

pois a regra de negócio mudou

Melhor NÃO se...

1. Jurado de morte
2. “Seria mais fácil reescrever tudo”
3. Entrega com prazo curto

O quê?

Quando?

Como?

Técnicas e Hands-on

OBJETIVO **Identificar e executar refactoring e redesign**

TÉCNICAS

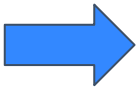
Cultura

“Não temos tempo para débito técnico, temos outras prioridades”

“Posso arremessar pedras pois já existem janelas quebradas.” (Teoria das Janelas Quebradas)

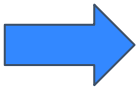
Cultura

“Não temos tempo para débito técnico, temos outras prioridades”



"Reserve 20% do ciclo de desenvolvimento para requisitos não-funcionais e redução de débito técnico" (DevOps Handbook)

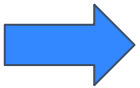
“Posso arremessar pedras pois já existem janelas quebradas.” (Teoria das Janelas Quebradas)



“Sempre deixe o acampamento mais limpo do que você encontrou.” (Lei dos Escoteiros)

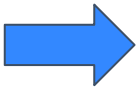
Cultura

“Não temos tempo para débito técnico, temos outras prioridades”



"Reserve 20% do ciclo de desenvolvimento para requisitos não-funcionais e redução de débito técnico" (DevOps Handbook)

“Posso arremessar pedras pois já existem janelas quebradas.” (Teoria das Janelas Quebradas)



“Sempre deixe o acampamento mais limpo do que você encontrou.” (Lei dos Escoteiros)

+ Mudanças rápidas

+ Contratação

– Incidentes

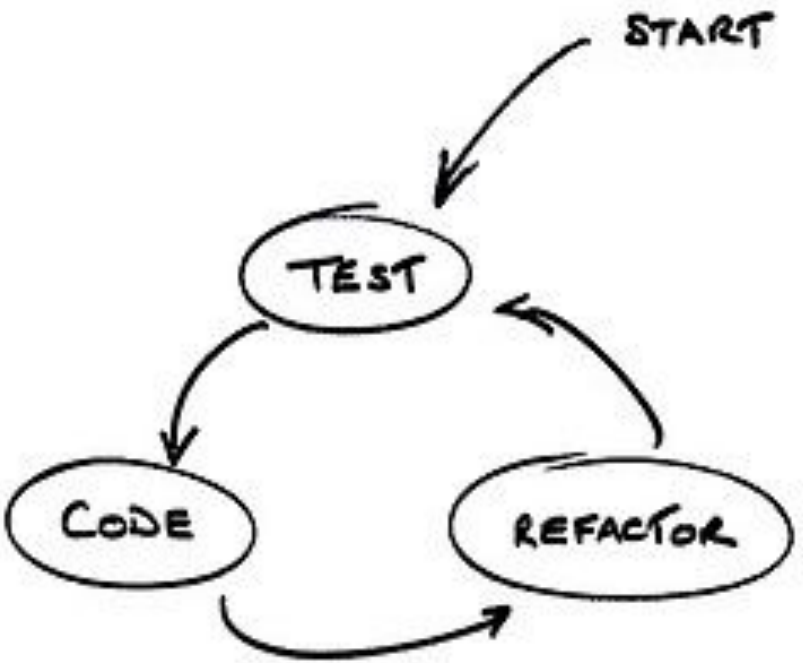
Como refatorar?

1. Testes
2. Métricas
3. Ferramentas

Como refatorar?

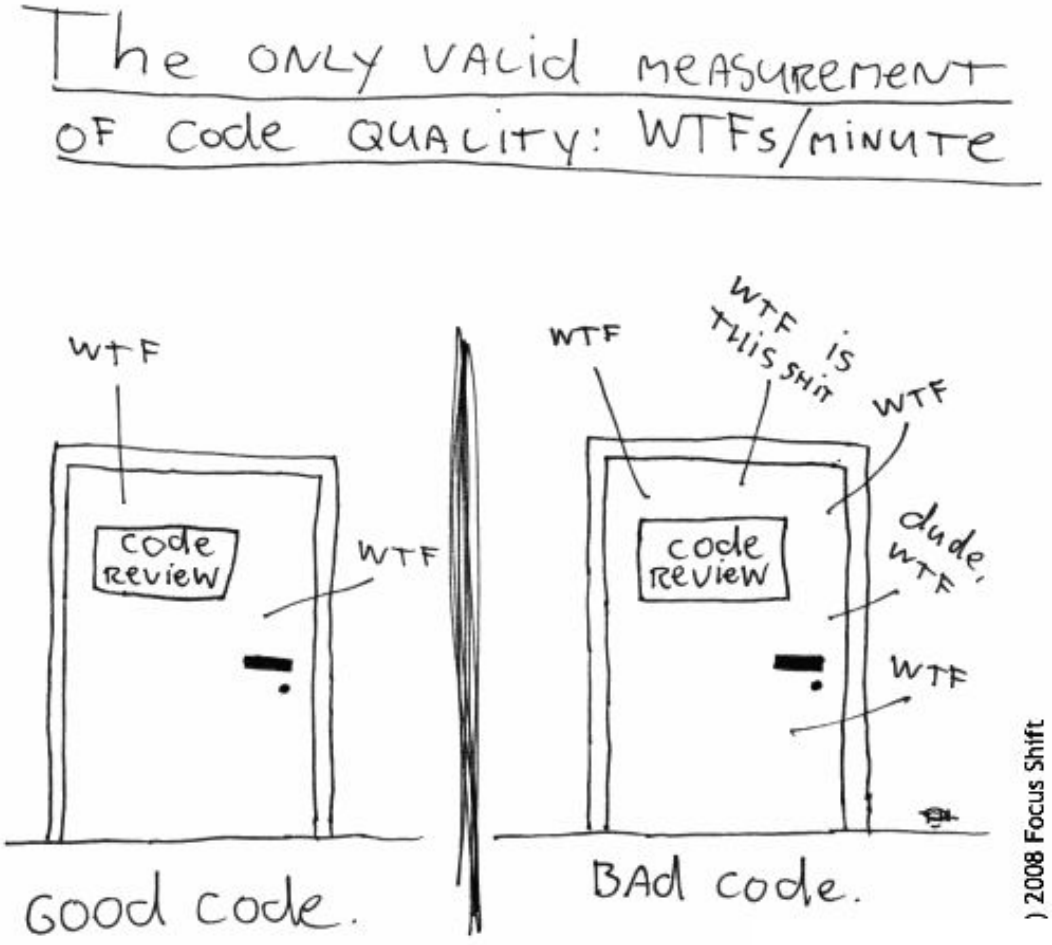
1. Testes

Garantir características não-funcionais



Como refatorar?

2. Métricas



Como refatorar?

2 Métricas

- Linhas de Código - LOC
- Número de Classes - NC
- Número de Componentes da Classe
- Profundidade da Árvore de Herança - DIT
- Número de Filhos - NOC
- Acoplamento Entre Objetos - CBO
- Conexões Aferentes por Classe - ACC
- Resposta de uma Classe - RFC
- Fator de Acoplamento - COF
- Ausência de Coesão em Métodos - LCOM4
- Complexidade Ciclomática

Como refatorar?

2. Métricas

ABC Metric

|ABC| = sqrt((A*A)+(B*B)+(C*C))

A: Assignment

Ex: = *= /= %= += <<= >>= &= |= ^= >>>= ++ --

B: Branch

Ex: chamada de função ou nova operação

C: Condition

Ex: == != <= >= < > else, case try, catch

Como refatorar?

2. Ferramentas

- Flog (Ruby)
- Analizo (Java, C, C++)
- Radon (Python)
- PHPMD (PHP)

- Mezuro
- Sonar
- Code Climate

Overview Progress Issues Code Trends Last master build 3 days ago Refresh

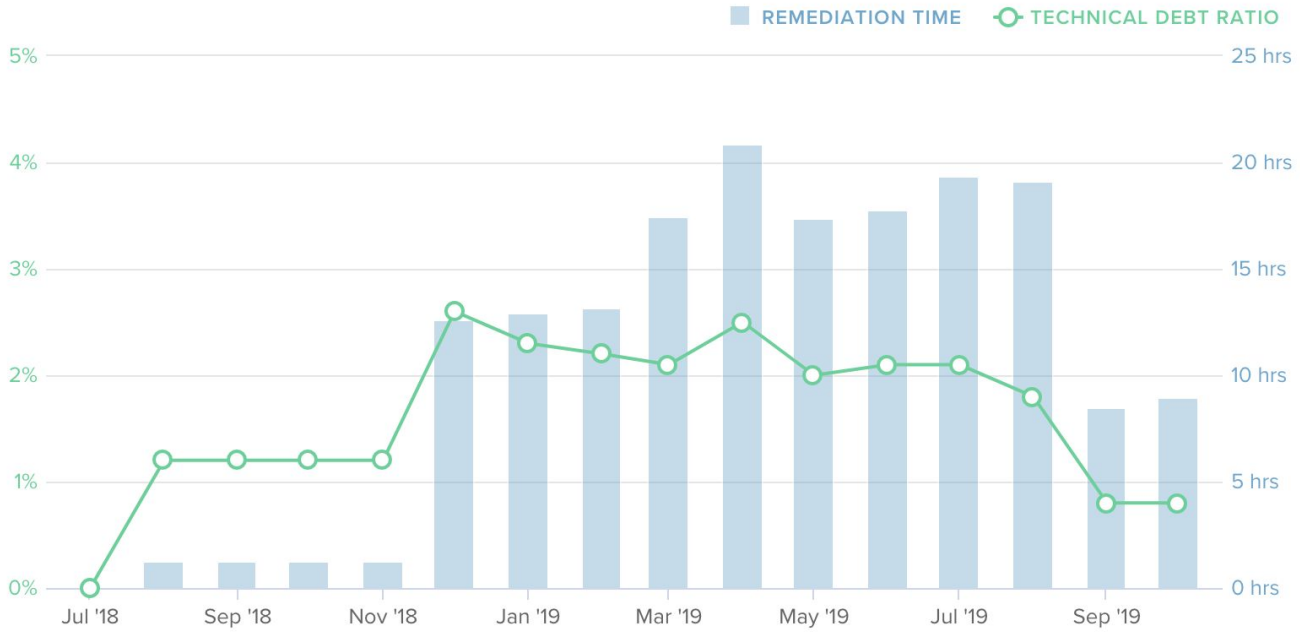
MAINTAINABILITY

- Technical debt
- Lines of code
- Churn vs. maintainability

TEST COVERAGE

- Overall coverage
- New code coverage

Technical Debt



* Code Climate: https://codeclimate.com

Hands-on Refactor

Refactor Hands-on
Nested conditionals

Existem 2 casos de refactor:

1. Simples
2. Complexo

Refactor Hands-on
Nested conditionals

Existem 2 casos de refactor:

1. Simples
2. Complexo

Refactor Hands-on
Nested conditionals

CÓDIGO

TESTES

GILTED ROSE KATA

Refactor Hands-on
Nested conditionals

CÓDIGO

TESTES

GildedRoseKata.java

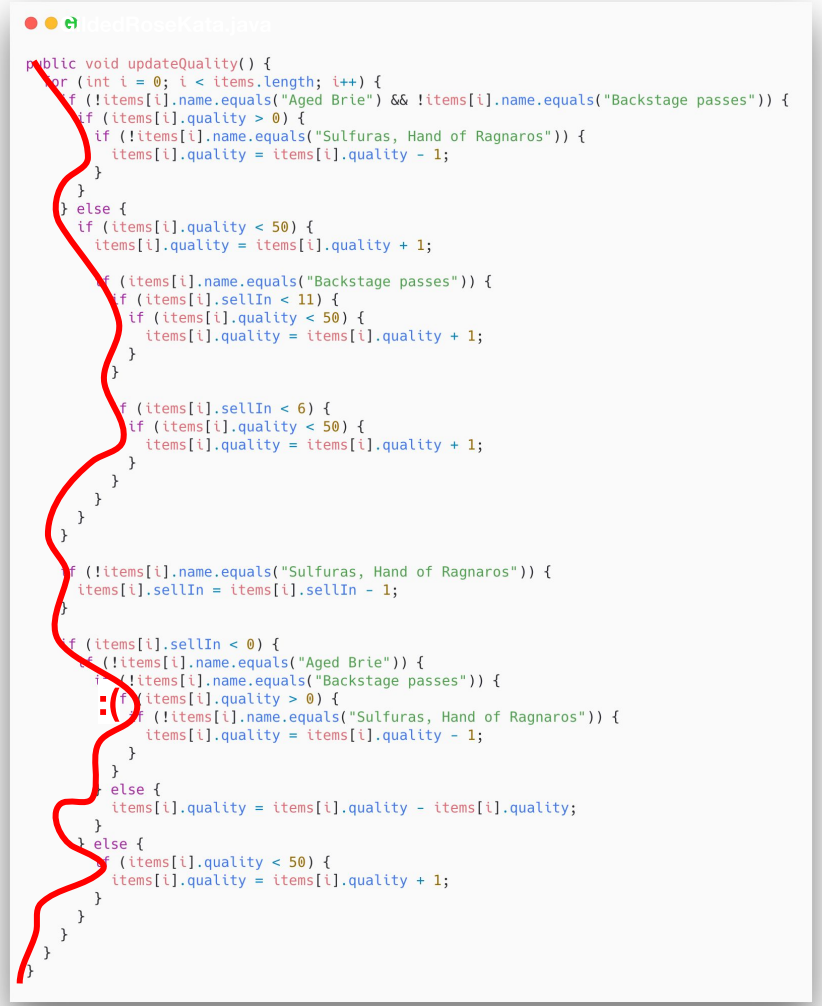
```
public void updateQuality() {
  for (int i = 0; i < items.length; i++) {
    if (!items[i].name.equals("Aged Brie") && !items[i].name.equals("Backstage passes")) {
      if (items[i].quality > 0) {
        if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
          items[i].quality = items[i].quality - 1;
        }
      }
    } else {
      if (items[i].quality < 50) {
        items[i].quality = items[i].quality + 1;

        if (items[i].name.equals("Backstage passes")) {
          if (items[i].sellIn < 11) {
            if (items[i].quality < 50) {
              items[i].quality = items[i].quality + 1;
            }
          }

          if (items[i].sellIn < 6) {
            if (items[i].quality < 50) {
              items[i].quality = items[i].quality + 1;
            }
          }
        }
      }
    }

    if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
      items[i].sellIn = items[i].sellIn - 1;
    }

    if (items[i].sellIn < 0) {
      if (!items[i].name.equals("Aged Brie")) {
        if (!items[i].name.equals("Backstage passes")) {
          if (items[i].quality > 0) {
            if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
              items[i].quality = items[i].quality - 1;
            }
          }
        } else {
          items[i].quality = items[i].quality - items[i].quality;
        }
      } else {
        if (items[i].quality < 50) {
          items[i].quality = items[i].quality + 1;
        }
      }
    }
  }
}
```

* Source: https://github.com/emilybache/GildedRose-Refactoring-Kata

Refactor Hands-on
Nested conditionals

CÓDIGO

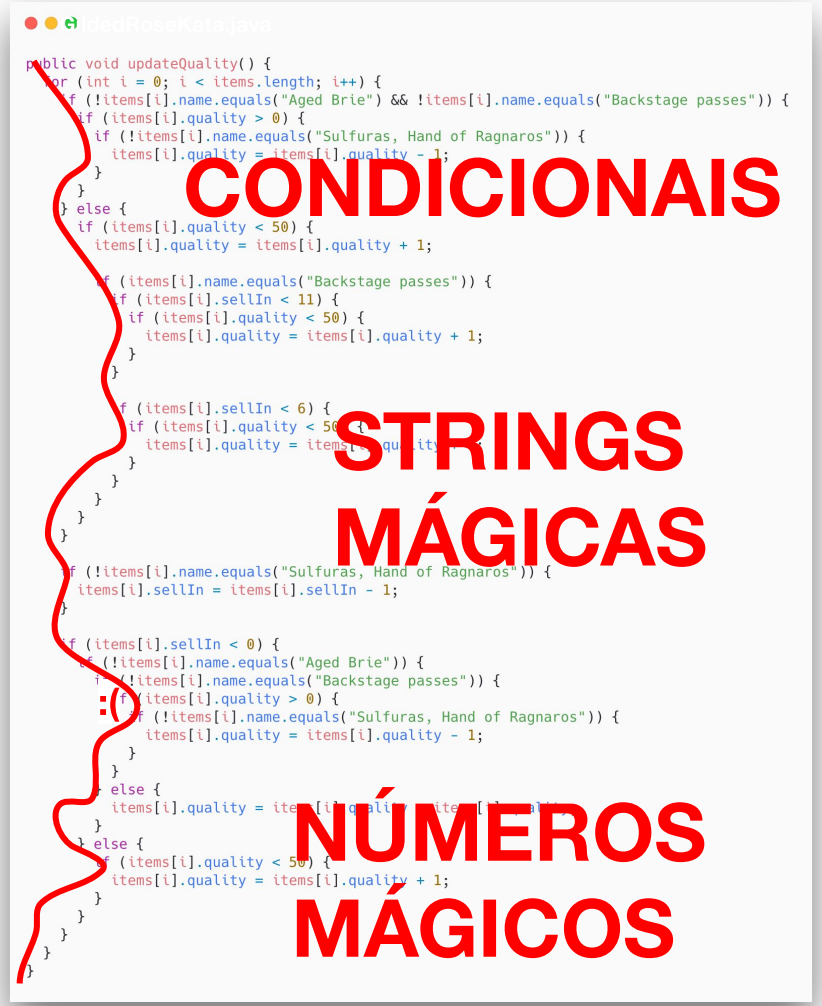
TESTES



Refactor Hands-on
Nested conditionals

CÓDIGO

TESTES



Refactor Hands-on
Nested conditionals

CÓDIGO

TESTES

Big Conditional

Complexidade: 50

```
public void updateQuality() {
  for (int i = 0; i < items.length; i++) {
    if (!items[i].name.equals("Aged Brie") && !items[i].name.equals("Backstage passes")) {
      if (items[i].quality > 0) {
        if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
          items[i].quality = items[i].quality - 1;
        }
      }
    } else {
      if (items[i].quality < 50) {
        items[i].quality = items[i].quality + 1;

        if (items[i].name.equals("Backstage passes")) {
          if (items[i].sellIn < 11) {
            if (items[i].quality < 50) {
              items[i].quality = items[i].quality + 1;
            }
          }

          if (items[i].sellIn < 6) {
            if (items[i].quality < 50) {
              items[i].quality = items[i].quality + 1;
            }
          }
        }
      }
    }

    if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
      items[i].sellIn = items[i].sellIn - 1;
    }

    if (items[i].sellIn < 0) {
      if (!items[i].name.equals("Aged Brie")) {
        if (!items[i].name.equals("Backstage passes")) {
          if (items[i].quality > 0) {
            if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
              items[i].quality = items[i].quality - 1;
            }
          }
        } else {
          items[i].quality = items[i].quality - items[i].quality;
        }
      } else {
        if (items[i].quality < 50) {
          items[i].quality = items[i].quality + 1;
        }
      }
    }
  }
}
```

Replace Magic Number with Symbolic Constant

Problem

Your code uses a number that has a certain meaning to it.

```
double potentialEnergy(double mass, doubl
  return mass * height * 9.81;
}
```

Solution

Replace this number with a constant that has a human-readable name explaining the meaning of the number.

```
static final double GRAVITATIONAL_CONSTAN

double potentialEnergy(double mass, doubl
  return mass * height * GRAVITATIONAL_CO
}
```

Replace Nested Conditional with Guard Clauses

Problem

You have a group of nested conditionals and it is hard to determine the normal flow of code execution.

```
public double getPayAmount() {
  double result;
  if (isDead){
    result = deadAmount();
  }
  else {
    if (isSeparated){
      result = separatedAmount();
    }
    else {
      if (isRetired){
        result = retiredAmount();
      }
      else{
        result = normalPayAmount();
      }
    }
  }
  return result;
}
```

Solution

Isolate all special checks and edge cases into separate clauses and place them before the main checks. Ideally, you should have a “flat” list of conditionals, one after the other.

```
public double getPayAmount() {
  if (isDead){
    return deadAmount();
  }
  if (isSeparated){
    return separatedAmount();
  }
  if (isRetired){
    return retiredAmount();
  }
  return normalPayAmount();
}
```

ace Magic Number with bolic Constant

Solution

a number that has a certain

Replace this number with a constant that has a human-readable name explaining the meaning of the number.

```
tialEnergy(double mass, doubl
s * height * 9.81;
```

```
static final double GRAVITATIONAL_CONSTAN

double potentialEnergy(double mass, doubl
  return mass * height * GRAVITATIONAL_CO
}
```

Replace Nested Conditio with Guard Clauses

Problem

You have a group of nested conditionals and it is hard to determine the normal flow of code execution.

```
public double getPayAmount() {
  double result;
  if (isDead){
    result = deadAmount();
  }
  else {
    if (isSeparated){
      result = separatedAmount();
    }
    else {
      if (isRetired){
        result = retiredAmount();
      }
      else{
        result = normalPayAmount();
      }
    }
  }
  return result;
}
```

Solution

Isolate all special checks separate clauses and pla main checks. Ideally, you list of conditionals, one

```
public double getPayA
  if (isDead){
    return deadAmount
  }
  if (isSeparated){
    return separatedA
  }
  if (isRetired){
    return retiredAmo
  }
  return normalPayAmo
}
```

Consolidate Conditional Expression

Problem

You have multiple conditionals that lead to the same result or action.

```
double disabilityAmount() {
  if (seniority < 2) {
    return 0;
  }
  if (monthsDisabled > 12) {
    return 0;
  }
  if (isPartTime) {
    return 0;
  }
  // Compute the disability amount.
  // ...
}
```

Solution

Consolidate all these conditionals in a single expression.

```
double disabilityAmount() {
  if (isNotEligableForDisability()) {
    return 0;
  }
  // Compute the disability amount.
  // ...
}
```

Replace Nested Conditio with Guard Clauses

Problem

You have a group of nested conditionals and it is hard to determine the normal flow of code execution.

```
public double getPayAmount() {
  double result;
  if (isDead){
    result = deadAmount();
  }
  else {
    if (isSeparated){
      result = separatedAmount();
    }
    else {
      if (isRetired){
        result = retiredAmount();
      }
      else{
        result = normalPayAmount();
      }
    }
  }
  return result;
}
```

Solution

Isolate all special checks separate clauses and pla main checks. Ideally, you list of conditionals, one

```
public double getPayA
  if (isDead){
    return deadAmount
  }
  if (isSeparated){
    return separatedA
  }
  if (isRetired){
    return retiredAmo
  }
  return normalPayAmo
}
```

Consolidate Conditional Expression

Problem

You have multiple conditionals that lead to the same result or action.

```
double disabil
  if (seniorit
    return 0;
  }
  if (monthsDi
    return 0;
  }
  if (isPartTi
    return 0;
  }
  // Compute t
  // ...
}
```

Solution

Consolidate all these conditionals in a single expression.

Decompose Conditional

Problem

You have a complex conditional (`if-then` / `else` or `switch`).

```
if (date.before(SUMMER_START) || date.aft
  charge = quantity * winterRate + winter
}
else {
  charge = quantity * summerRate;
}
```

Solution

Decompose the complicated parts of the conditional into separate methods: the condition, `then` and `else` .

```
if (isSummer(date)) {
  charge = summerCharge(quantity);
}
else {
  charge = winterCharge(quantity);
}
```

Repla with

Problem

You have a gro
it is hard to de
code executior

```
public doubl
  double res
  if (isDeac
    result =
  }
  else {
    if (isSe
      result
    }
    else {
      if (is
        resu
      }
      else{
        resu
      }
    }
  }
  return res
}
```

Replace Conditional with Polymorphism

Problem

You have a conditional that performs various actions depending on object type or properties.

```
class Bird {
  // ...
  double getSpeed() {
    switch (type) {
      case EUROPEAN:
        return getBaseSpeed();
      case AFRICAN:
        return getBaseSpeed() - getLoadFa
      case NORWEGIAN_BLUE:
        return (isNailed) ? 0 : getBaseSp
    }
    throw new RuntimeException("Should be
  }
}
```

Solution

Create subclasses matching the branches of the conditional. In them, create a shared method and move code from the corresponding branch of the conditional to it. Then replace the conditional with the relevant method call. The result is that the proper implementation will be attained via polymorphism depending on the object class.

```
abstract class Bird {
  // ...
  abstract double getSpeed();
}

class European extends Bird {
  double getSpeed() {
    return getBaseSpeed();
  }
}
class African extends Bird {
  double getSpeed() {
    return getBaseSpeed() - getLoadFactor
  }
}
class NorwegianBlue extends Bird {
  double getSpeed() {
    return (isNailed) ? 0 : getBaseSpeed(
  }
}
```

ate Conditional on

Solution

onals that lead to

Consolidate all these conditionals in a single expression.

```
ate.before(SUMMER_START) || date.aft
rge = quantity * winterRate + winter

{
rge = quantity * summerRate;
```

```
if (isSummer(date)) {
  charge = summerCharge(quantity);
}
else {
  charge = winterCharge(quantity);
}
```

ecompose Conditional

lem

e a complex conditional
n / `else` or `switch`).

Solution

Decompose the complicated parts of the conditional into separate methods: the condition, `then` and `else` .

Repla with

Problem

You have a gro
it is hard to de
code executior

```
public doubl
  double res
  if (isDeac
    result =
  }
  else {
    if (isSe
      result
    }
    else {
      if (is
        resu
      }
      else{
        resu
      }
    }
  }
  return res
}
```

Replace Conditional with Polymorphism

Problem

You have a condition
actions depending or
properties.

```
class Bird {
  // ...
  double getSpeed(
    switch (type)
      case EUROPEA
        return get
      case AFRICAN
        return get
      case NORWEGI
        return (is
    }
    throw new Runt
  }
}
```

```
class NorwegianBlue extends Bird {
  double getSpeed() {
    return (isNailed) ? 0 : getBaseSpeed(
  }
}
```

ate Conditional

tion

ate all these conditionals in a single
on.

Conditional

Solution

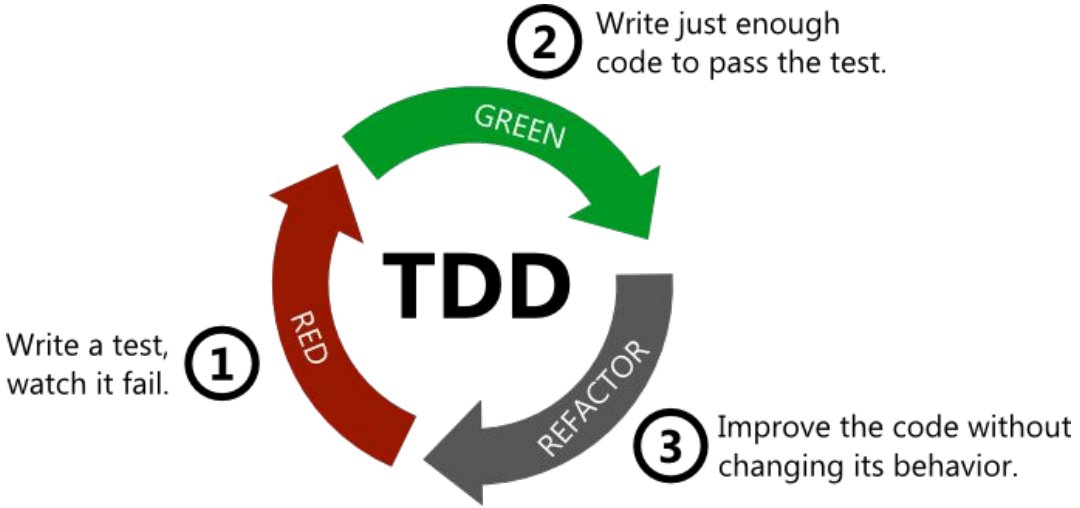
Decompose the complicated parts of the conditional into separate methods: the condition, `then` and `else`.

```
if (isSummer(date)) {
  charge = summerCharge(quantity);
}
else {
  charge = winterCharge(quantity);
}
```

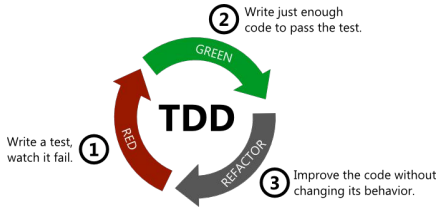
Refactor Hands-on
Nested conditionals

CÓDIGO

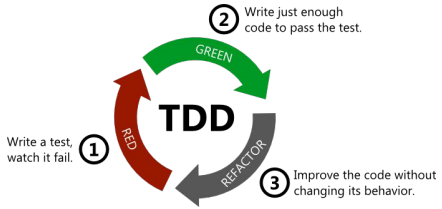
TESTES



```
class GildedRoseItem {

  public String name;
  public int quality;
  public int sellIn;

  public Item(String name, int quality, int sellIn) {
    this.name = name;
    this.quality = quality;
    this.sellIn = sellIn;
  }

  public void process() {
    switch(this.name) {
      case "Aged Brie":
        processBrie();
      case "Sulfuras, Hand of Ragnaros":
        processSulfuras();
      case "Backstage passes":
        processBackstage();
      default:
        processNormal();
    }
  }
}
```

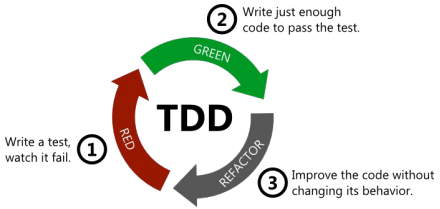


```
class GildedRoseItem {

  public String name;
  public int quality;
  public int sellIn;

  public Item(String name, int quality, int sellIn) {
    this.name = name;
    this.quality = quality;
    this.sellIn = sellIn;
  }

  public void process() {
    switch(this.name) {
      case "Aged Brie":
        processBrie();
      case "Sulfuras, Hand of Ragnaros":
        processSulfuras();
      case "Backstage passes":
        processBackstage();
      default:
        processNormal();
    }
  }
}
```




```
@Test
public void foo() {
  GildedRoseItem item = new GildedRoseItem("normal", 0, 5);
  item.process();
  assertEquals(0, item.quality);
  assertEquals(4, item.sellIn);
}
```

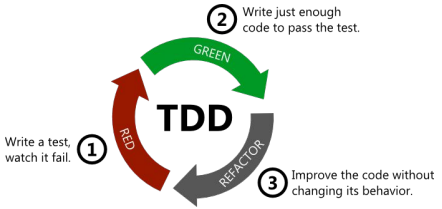
```
class GildedRoseItem {

  public String name;
  public int quality;
  public int sellIn;

  public Item(String name, int quality, int sellIn) {
    this.name = name;
    this.quality = quality;
    this.sellIn = sellIn;
  }

  public void processNormal() {

  }

  public void process() {
    switch(this.name) {
      case "Aged Brie":
        processBrie();
      case "Sulfuras, Hand of Ragnaros":
        processSulfuras();
      case "Backstage passes":
        processBackstage();
      default:
        processNormal();
    }
  }
}
```




```
@Test
public void foo() {
  GildedRoseItem item = new GildedRoseItem("normal", 0, 5);
  item.process();
  assertEquals(0, item.quality);
  assertEquals(4, item.sellIn);
}
```

Escrever método para caso de item "normal"

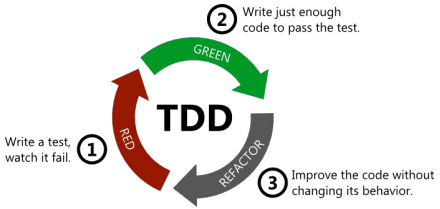
```
class GildedRoseItem {

  public String name;
  public int quality;
  public int sellIn;

  public Item(String name, int quality, int sellIn) {
    this.name = name;
    this.quality = quality;
    this.sellIn = sellIn;
  }

  public void processNormal() {

  }

  public void process() {
    switch(this.name) {
      case "Aged Brie":
        processBrie();
      case "Sulfuras, Hand of Ragnaros":
        processSulfuras();
      case "Backstage passes":
        processBackstage();
      default:
        processNormal();
    }
  }
}
```




```
@Test
public void foo() {
  GildedRoseItem item = new GildedRoseItem("normal", 0, 5);
  item.process();
  assertEquals(0, item.quality);
  assertEquals(4, item.sellIn);
}
```

Testes falham

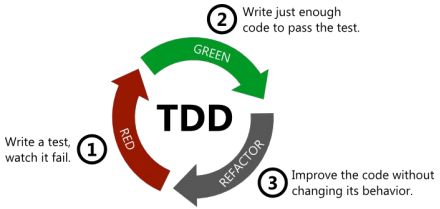
```
public Item(String name, int quality, int sellIn) {
  this.name = name;
  this.quality = quality;
  this.sellIn = sellIn;
}

public void processNormal() {
  if (quality != 0) {
    if (sellIn > 0) {
      quality = quality - 1;
    }
    if (sellIn <= 0) {
      quality = quality - 2;
    }
  }

  sellIn = sellIn - 1;
}

public void process() {
  switch(this.name) {
    case "Aged Brie":
      processBrie();
    case "Sulfuras, Hand of Ragnaros":
      processSulfuras();
    case "Backstage passes":
      processBackstage();
    default:
      processNormal();
  }
```




```
@Test
public void foo() {
  GildedRoseItem item = new GildedRoseItem("normal", 0, 5);
  item.process();
  assertEquals(0, item.quality);
  assertEquals(4, item.sellIn);
}
```

Fazer o teste passar, com código bruto

```
public Item(String name, int quality, int sellIn) {
  this.name = name;
  this.quality = quality;
  this.sellIn = sellIn;
}

public void processNormal() {
  if (quality != 0) {
    if (sellIn > 0) {
      quality = quality - 1;
    }
    if (sellIn <= 0) {
      quality = quality - 2;
    }
  }

  sellIn = sellIn - 1;
}

public void process() {
  switch(this.name) {
    case "Aged Brie":
      processBrie();
    case "Sulfuras, Hand of Ragnaros":
      processSulfuras();
    case "Backstage passes":
      processBackstage();
    default:
      processNormal();
  }
```




```
@Test
public void foo() {
  GildedRoseItem item = new GildedRoseItem("normal", 0, 5);
  item.process();
  assertEquals(0, item.quality);
  assertEquals(4, item.sellIn);
}
```

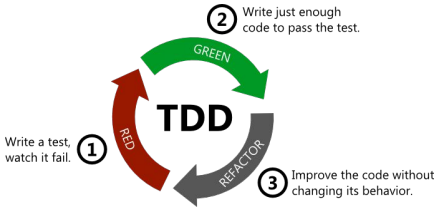
Testes passam

Agora sim, refatorar!

```
public Item(String name, int quality, int sellIn) {
  this.name = name;
  this.quality = quality;
  this.sellIn = sellIn;
}
```

```
public void processNormal() {
  sellIn -= 1;
  if (quality == 0) return;
  quality -= 1;
  if (sellIn <= 0) quality -= 1;
}
```

```
public void process() {
  switch(this.name) {
    case "Aged Brie":
      processBrie();
    case "Sulfuras, Hand of Ragnaros":
      processSulfuras();
    case "Backstage passes":
      processBackstage();
    default:
      processNormal();
  }
}
```



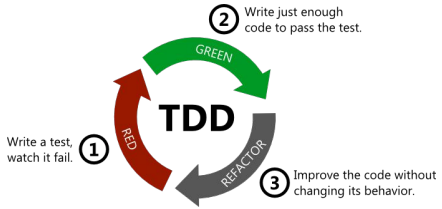

```
@Test
public void foo() {
  GildedRoseItem item = new GildedRoseItem("normal", 0, 5);
  item process();
  assertEquals(0, item quality);
  assertEquals(4, item sellIn);
}
```

Refactor done :)

```
public Item(String name, int quality, int sellIn) {
  this.name = name;
  this.quality = quality;
  this.sellIn = sellIn;
}
```

```
public void processNormal() {
  sellIn -= 1;
  if (quality == 0) return;
  quality -= 1;
  if (sellIn <= 0) quality -= 1;
}
```

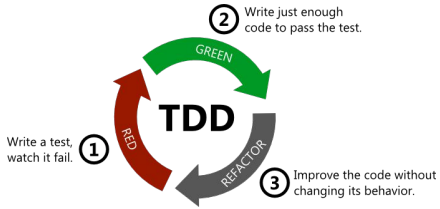
```
public void process() {
  switch(this.name) {
    case "Aged Brie":
      processBrie();
    case "Sulfuras, Hand of Ragnaros":
      processSulfuras();
    case "Backstage passes":
      processBackstage();
    default:
      processNormal();
  }
}
```




```
@Test
public void foo() {
  GildedRoseItem item = new GildedRoseItem("normal", 0, 5);
  item.process();
  assertEquals(0, item.quality);
  assertEquals(4, item.sellIn);
}
```

Testes passam com refactor

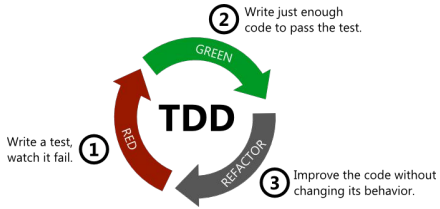
```
public void processNormal() {
  sellIn -= 1;
  if (quality == 0) return;
  quality -= 1;
  if (sellIn <= 0) quality -= 1;
}

public void processBrie() {
  sellIn -= 1;
  if (quality >= 50) return;
  quality += 1;
  if (sellIn <= 0) quality += 1;
}

public void processSulfuras() {

}

public void processBackstage() {
  sellIn -= 1;
  if (quality >= 50) return;
  if (sellIn < 0) quality = 0;

  quality += 1;
  if (sellIn < 10) quality += 1;
  if (sellIn < 5) quality += 1;
}

public void process() {
  switch(this.name) {
    case "Aged Brie":
      processBrie();
    case "Sulfuras, Hand of Ragnaros":
      processSulfuras();
    case "Backstage passes":
      processBackstage();
     default:
       processNormal();
  }
}
```



Repetir para os outros 3 casos

*Testes do Sulfuras passaram sem nenhum código - TDD permitiu

```
public void processNormal() {
  sellIn -= 1;
  if (quality == 0) return;
  quality -= 1;
  if (sellIn <= 0) quality -= 1;
}

public void processBrie() {
  sellIn -= 1;
  if (quality >= 50) return;
  quality += 1;
  if (sellIn <= 0) quality += 1;
}

public void processSulfuras() {

}

public void processBackstage() {
  sellIn -= 1;
  if (quality >= 50) return;
  if (sellIn < 0) quality = 0;

  quality += 1;
  if (sellIn < 10) quality += 1;
  if (sellIn < 5) quality += 1;
}

public void process() {
  switch(this.name) {
    case "Aged Brie":
      processBrie();
    case "Sulfuras, Hand of Ragnaros":
      processSulfuras();
    case "Backstage passes":
      processBackstage();
    default:
      processNormal();
  }
}
```



Testes do Sulfuras passaram sem nenhum código - TDD permitiu perceber isso!

Ao ver o problema, é um item raro que não é vendido, logo, não precisa ser atualizado

```java
class GildedRoseItem {

  public String name;
  public int quality;
  public int sellIn;

  public Item(String name, int quality, int sellIn) {
    this.name = name;
    this.quality = quality;
    this.sellIn = sellIn;
  }

  public void processNormal() {
    sellIn -= 1;
    if (quality == 0) return;
    quality -= 1;
    if (sellIn <= 0) quality -= 1;
  }

  public void processBrie() {
    sellIn -= 1;
    if (quality >= 50) return;
    quality += 1;
    if (sellIn <= 0) quality += 1;
  }

  public void processSulfuras() {

  }

  public void processBackstage() {
    sellIn -= 1;
    if (quality >= 50) return;
    if (sellIn < 0) quality = 0;

    quality += 1;
    if (sellIn < 10) quality += 1;
    if (sellIn < 5) quality += 1;
  }

  public void process() {
    switch(this.name) {
      case "Aged Brie":
        processBrie();
      case "Sulfuras, Hand of Ragnaros":
        processSulfuras();
      case "Backstage passes":
        processBackstage();
      default:
        processNormal();
    }
  }
}
```

Small methods

Complexidade: 40

Big Conditional

50

84

```
public void processNormal() {
  if (this.quality != 0) {
    if (this.sellIn > 0) {
      this.quality = this.quality - 1;
    }
    if (this.sellIn <= 0) {
      this.quality = this.quality - 2;
    }
  }

  this.sellIn = this.sellIn - 1;
}
```

40

```
public void processNormal() {
  sellIn -= 1;
  if (quality == 0) return;
  quality -= 1;
  if (sellIn <= 0) quality -= 1;
}
```

Big Conditional

50

```
public void processNormal() {
  if (this.quality != 0) {
    if (this.sellIn > 0) {
      this.quality = this.quality - 1;
    }
    if (this.sellIn <= 0) {
      this.quality = this.quality - 2;
    }
  }

  this.sellIn = this.sellIn - 1;
}
```

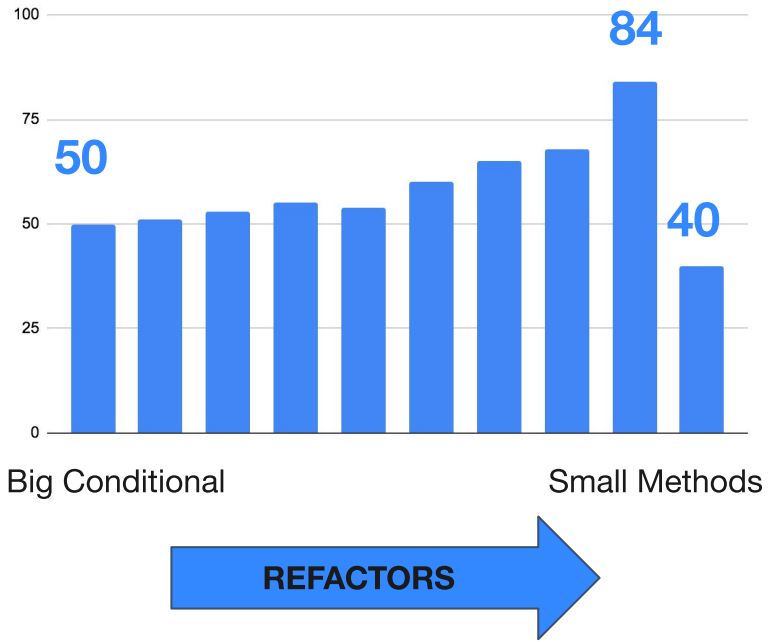
84

```
public void processNormal() {
  sellIn -= 1;
  if (quality == 0) return;
  quality -= 1;
  if (sellIn <= 0) quality -= 1;
}
```

40

Refactor não é sempre reduzir a complexidade a cada passo

Confie no processo!



Small Method 40

```
public void processNormal() {
  sellIn -= 1;
  if (quality == 0) return;
  quality -= 1;
  if (sellIn <= 0) quality -= 1;
}
```

É possível melhorar!

Small Method

```
public void processNormal() {
  sellIn -= 1;
  if (quality == 0) return;
  quality -= 1;
  if (sellIn <= 0) quality -= 1;
}
```

Small Object

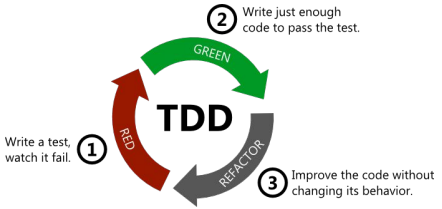
```
public class Normal extends Item {
  public void process() {
    sellIn -= 1;
    if (quality == 0) return;
    quality -= 1;
    if (sellIn <= 0) quality -= 1;
  }
}
```

Small Method

```
public void processNormal() {
  sellIn -= 1;
  if (quality == 0) return;
  quality -= 1;
  if (sellIn <= 0) quality -= 1;
}
```

Small Object

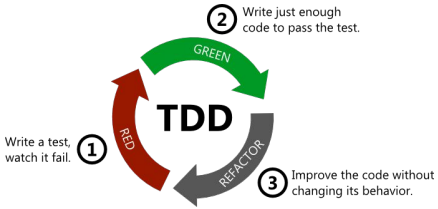
```
public class Normal extends Item {
  public void process() {
    sellIn -= 1;
    if (quality == 0) return;
    quality -= 1;
    if (sellIn <= 0) quality -= 1;
  }
}
```




```
@Test
public void foo() {
  GildedRoseItem item = new GildedRoseItem("normal", 0, 5);
  item.process();
  assertEquals(0, item.quality);
  assertEquals(4, item.sellIn);
}
```

Testes passam com refactor

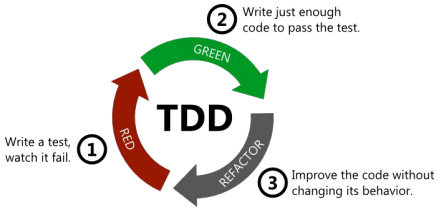
```
module GildedRose {
  public class Item {
    public String name;
    public int sellIn;
    public int quality;

    public Item(String name, int sellIn, int quality) {
      this.name = name;
      this.sellIn = sellIn;
      this.quality = quality;
    }

    public void process();
  }

  public class Normal extends Item {
    public void process() {
      sellIn -= 1;
      if (quality == 0) return;
      quality -= 1;
      if (sellIn <= 0) quality -= 1;
    }
  }

  public class Brie extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      quality += 1;
      if (sellIn <= 0) quality += 1;
    }
  }

  public class Backstage extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      if (sellIn < 0) quality = 0;

      quality += 1;
      if (sellIn < 10) quality += 1;
      if (sellIn < 5) quality += 1;
    }
  }
}
```



Repetir para os outros casos

Transformando cada small method em small object

```
module GildedRose {
  public class Item {
    public String name;
    public int sellIn;
    public int quality;

    public Item(String name, int sellIn, int quality) {
      this.name = name;
      this.sellIn = sellIn;
      this.quality = quality;
    }

    public void process();
  }

  public class Normal extends Item {
    public void process() {
      sellIn -= 1;
      if (quality == 0) return;
      quality -= 1;
      if (sellIn <= 0) quality -= 1;
    }
  }

  public class Brie extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      quality += 1;
      if (sellIn <= 0) quality += 1;
    }
  }

  public class Backstage extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      if (sellIn < 0) quality = 0;

      quality += 1;
      if (sellIn < 10) quality += 1;
      if (sellIn < 5) quality += 1;
    }
  }
}
```



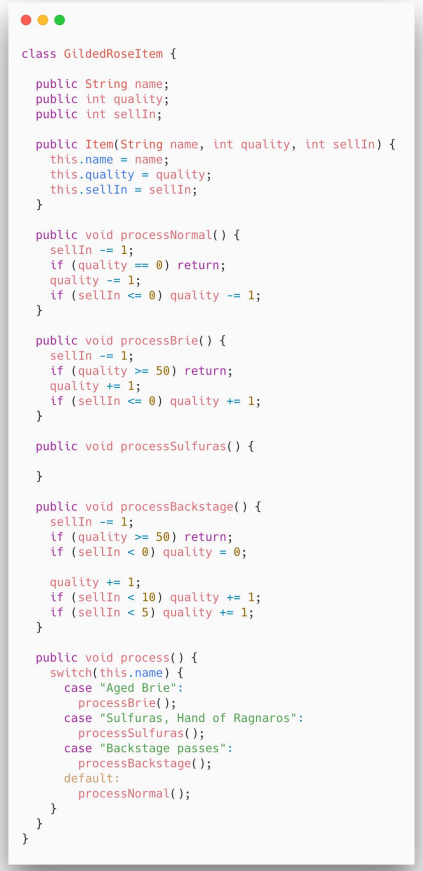
Small objects

Complexidade: 33

```
public void updateQuality() {
  for (int i = 0; i < items.length; i++) {
    if (!items[i].name.equals("Aged Brie") && !items[i].name.equals("Backstage passes")) {
      if (items[i].quality > 0) {
        if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
          items[i].quality = items[i].quality - 1;
        }
      }
    } else {
      if (items[i].quality < 50) {
        items[i].quality = items[i].quality + 1;

        if (items[i].name.equals("Backstage passes")) {
          if (items[i].sellIn < 11) {
            if (items[i].quality < 50) {
              items[i].quality = items[i].quality + 1;
            }
          }

          if (items[i].sellIn < 6) {
            if (items[i].quality < 50) {
              items[i].quality = items[i].quality + 1;
            }
          }
        }
      }
    }

    if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
      items[i].sellIn = items[i].sellIn - 1;
    }

    if (items[i].sellIn < 0) {
      if (!items[i].name.equals("Aged Brie")) {
        if (!items[i].name.equals("Backstage passes")) {
          if (items[i].quality > 0) {
            if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
              items[i].quality = items[i].quality - 1;
            }
          }
        } else {
          items[i].quality = items[i].quality - items[i].quality;
        }
      } else {
        if (items[i].quality < 50) {
          items[i].quality = items[i].quality + 1;
        }
      }
    }
  }
}
```

Big Conditional

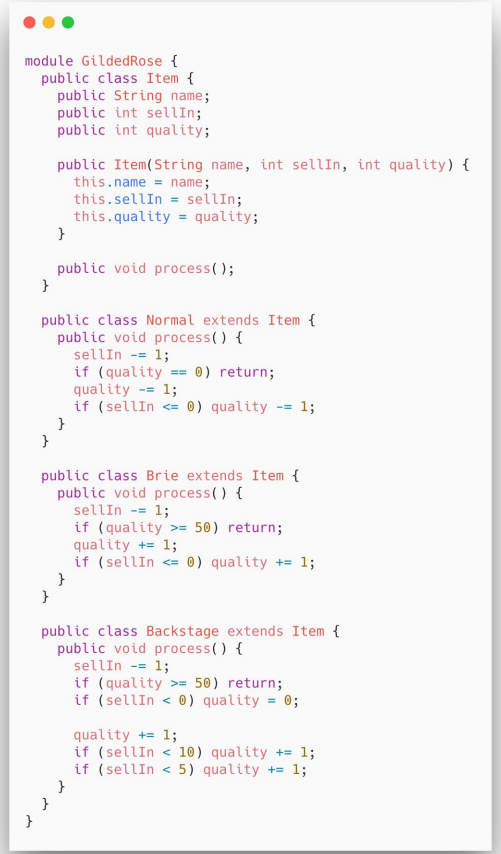
Complexidade: 50



```
class GildedRoseItem {

  public String name;
  public int quality;
  public int sellIn;

  public Item(String name, int quality, int sellIn) {
    this.name = name;
    this.quality = quality;
    this.sellIn = sellIn;
  }

  public void processNormal() {
    sellIn -= 1;
    if (quality == 0) return;
    quality -= 1;
    if (sellIn <= 0) quality -= 1;
  }

  public void processBrie() {
    sellIn -= 1;
    if (quality >= 50) return;
    quality += 1;
    if (sellIn <= 0) quality += 1;
  }

  public void processSulfuras() {

  }

  public void processBackstage() {
    sellIn -= 1;
    if (quality >= 50) return;
    if (sellIn < 0) quality = 0;

    quality += 1;
    if (sellIn < 10) quality += 1;
    if (sellIn < 5) quality += 1;
  }

  public void process() {
    switch(this.name) {
      case "Aged Brie":
        processBrie();
      case "Sulfuras, Hand of Ragnaros":
        processSulfuras();
      case "Backstage passes":
        processBackstage();
      default:
        processNormal();
    }
  }
}
```

Small Methods

Complexidade: 40



```
module GildedRose {
  public class Item {
    public String name;
    public int sellIn;
    public int quality;

    public Item(String name, int sellIn, int quality) {
      this.name = name;
      this.sellIn = sellIn;
      this.quality = quality;
    }

    public void process();
  }

  public class Normal extends Item {
    public void process() {
      sellIn -= 1;
      if (quality == 0) return;
      quality -= 1;
      if (sellIn <= 0) quality -= 1;
    }
  }

  public class Brie extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      quality += 1;
      if (sellIn <= 0) quality += 1;
    }
  }

  public class Backstage extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      if (sellIn < 0) quality = 0;

      quality += 1;
      if (sellIn < 10) quality += 1;
      if (sellIn < 5) quality += 1;
    }
  }
}
```

Small objects

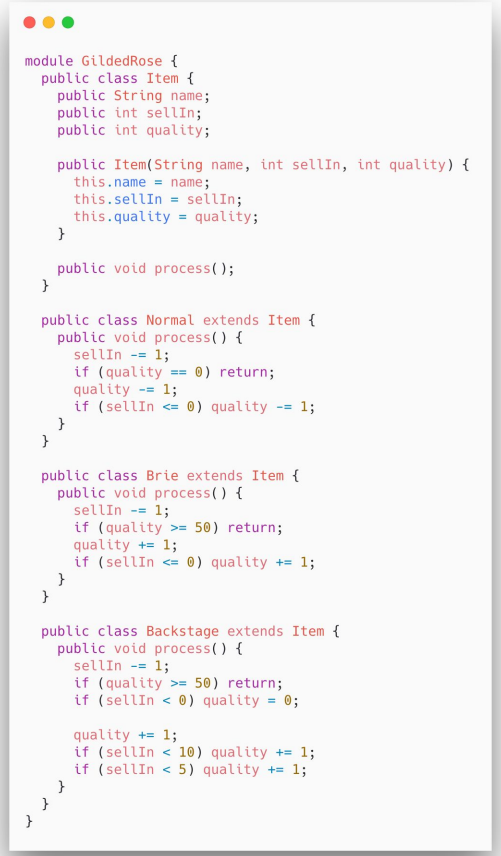
Complexidade: 33

Complexidade média:

7

Maior complexidade

12



```
module GildedRose {
  public class Item {
    public String name;
    public int sellIn;
    public int quality;

    public Item(String name, int sellIn, int quality) {
      this.name = name;
      this.sellIn = sellIn;
      this.quality = quality;
    }

    public void process();
  }

  public class Normal extends Item {
    public void process() {
      sellIn -= 1;
      if (quality == 0) return;
      quality -= 1;
      if (sellIn <= 0) quality -= 1;
    }
  }

  public class Brie extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      quality += 1;
      if (sellIn <= 0) quality += 1;
    }
  }

  public class Backstage extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      if (sellIn < 0) quality = 0;

      quality += 1;
      if (sellIn < 10) quality += 1;
      if (sellIn < 5) quality += 1;
    }
  }
}
```

Small objects

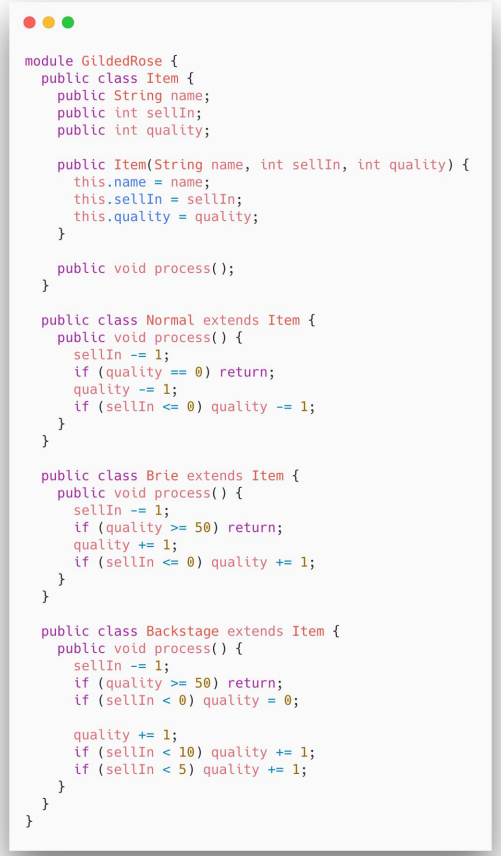
Complexidade: 33

Complexidade média:

7

Maior complexidade

12



```
module GildedRose {
  public class Item {
    public String name;
    public int sellIn;
    public int quality;

    public Item(String name, int sellIn, int quality) {
      this.name = name;
      this.sellIn = sellIn;
      this.quality = quality;
    }

    public void process();
  }

  public class Normal extends Item {
    public void process() {
      sellIn -= 1;
      if (quality == 0) return;
      quality -= 1;
      if (sellIn <= 0) quality -= 1;
    }
  }

  public class Brie extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      quality += 1;
      if (sellIn <= 0) quality += 1;
    }
  }

  public class Backstage extends Item {
    public void process() {
      sellIn -= 1;
      if (quality >= 50) return;
      if (sellIn < 0) quality = 0;

      quality += 1;
      if (sellIn < 10) quality += 1;
      if (sellIn < 5) quality += 1;
    }
  }
}
```

Small objects

Complexidade: 33

Hands-on Redesign

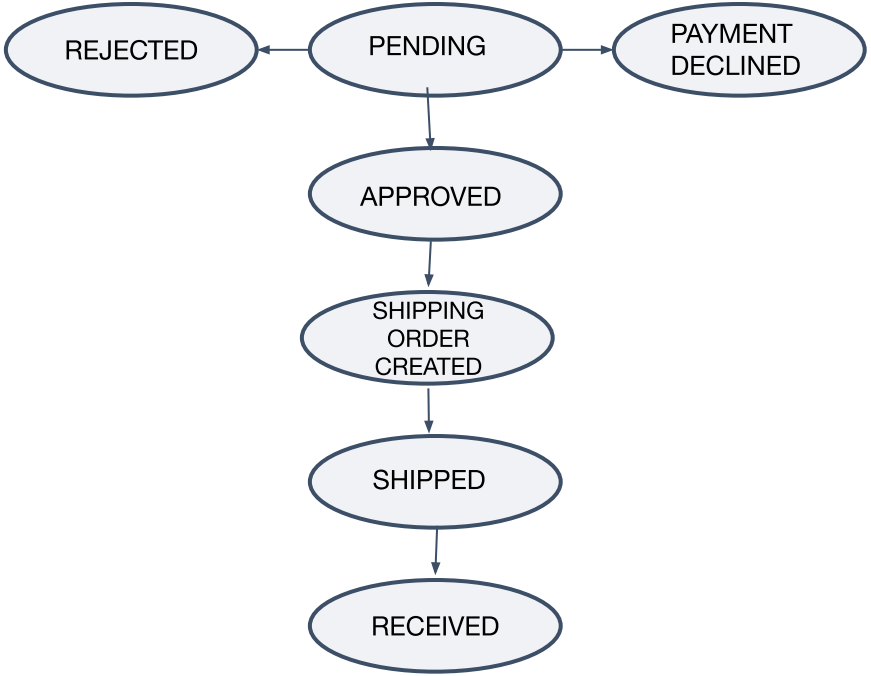
Como fazer um redesign?

1. Identificar a necessidade
2. Revisitar o problema
3. Desenhar a nova solução
4. Implementar
    - Codar nova solução
    - Resolver casos antigos
    - Excluir código da solução antiga

Redesign Hands-on State Machine

1. Identificar a necessidade

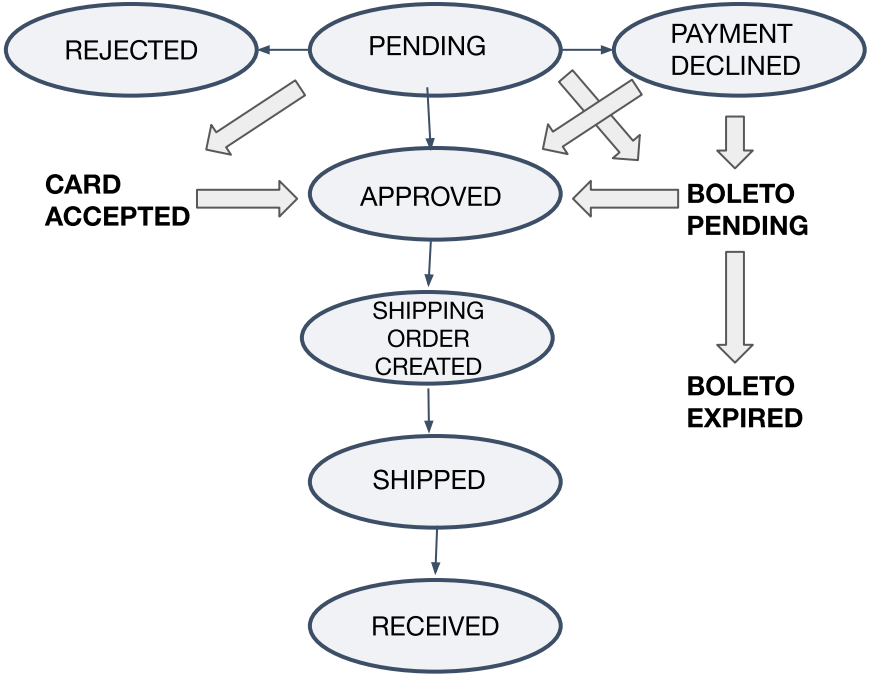
Nova demanda: Pagamento



Redesign Hands-on State Machine

1. Identificar a necessidade

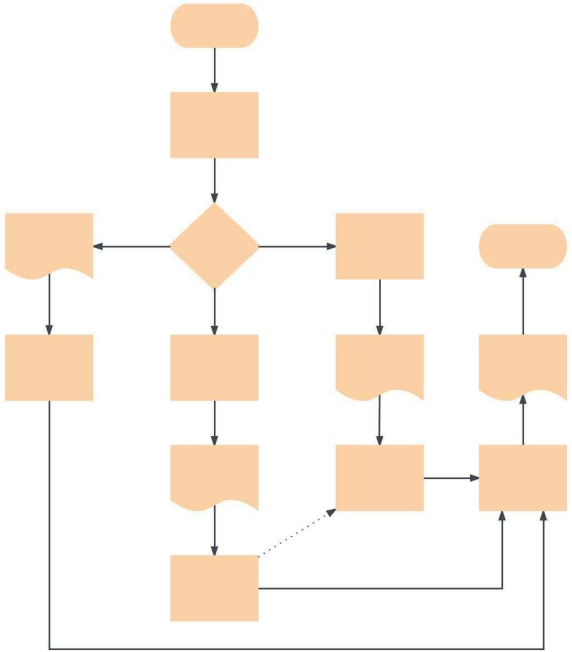
Nova demanda: Pagamento



Redesign Hands-on State Machine

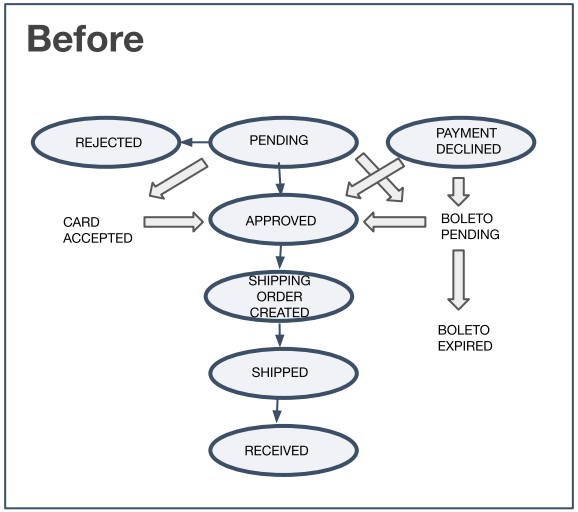
2 Revisitar o problema

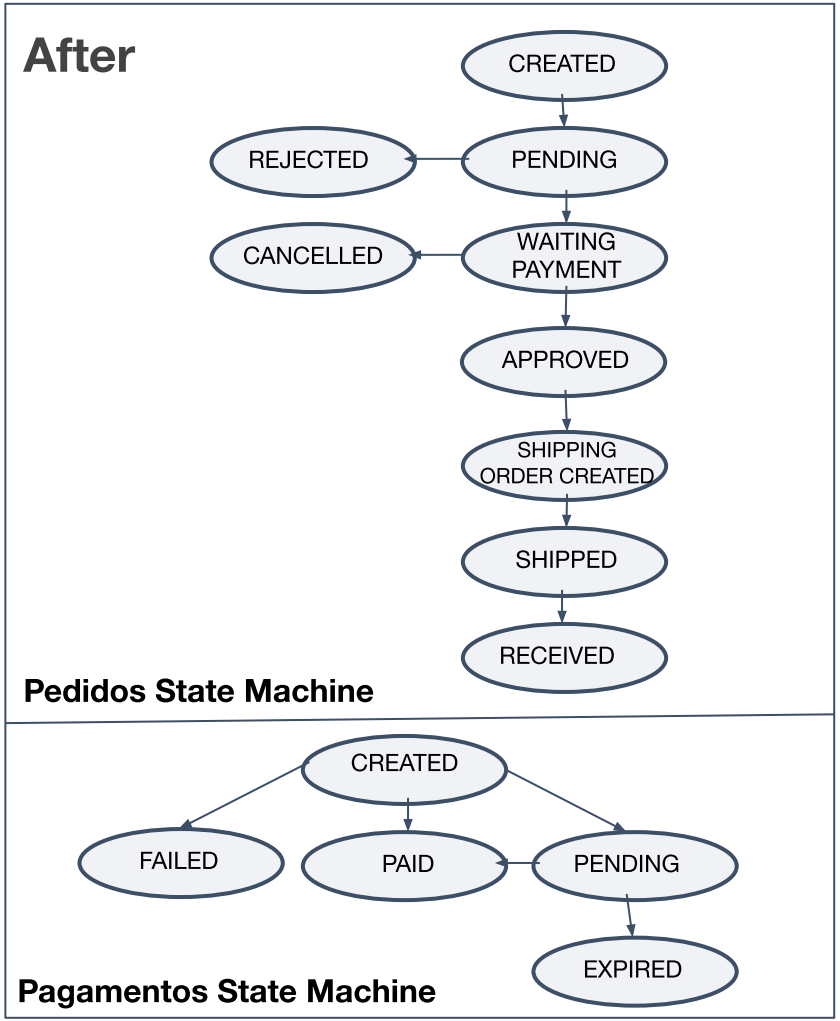
Entender o novo fluxo de pedidos



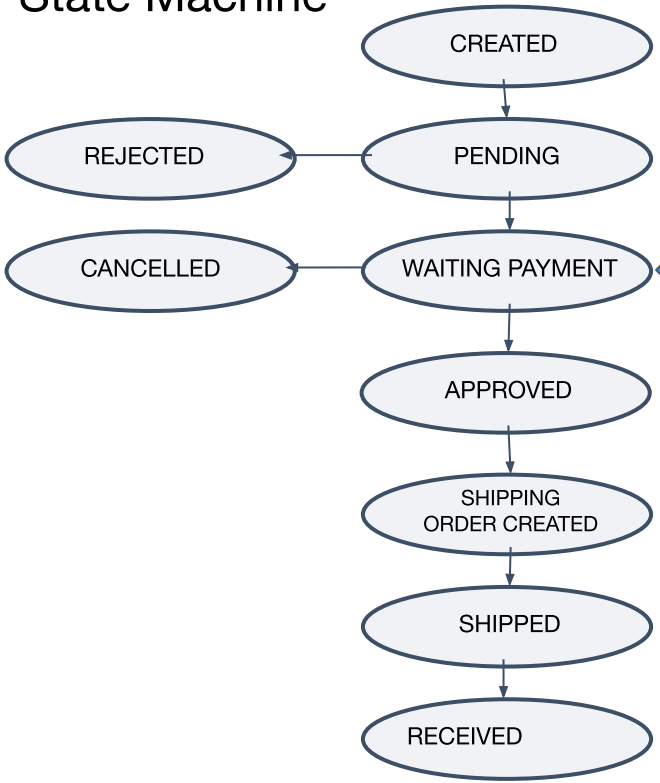
Redesign Hands-on State Machine

3 Desenhar a nova solução

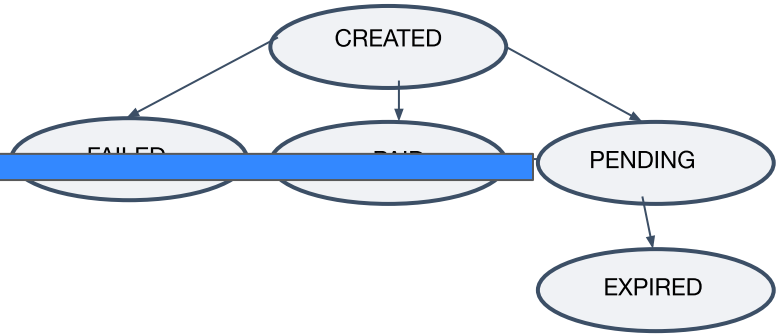




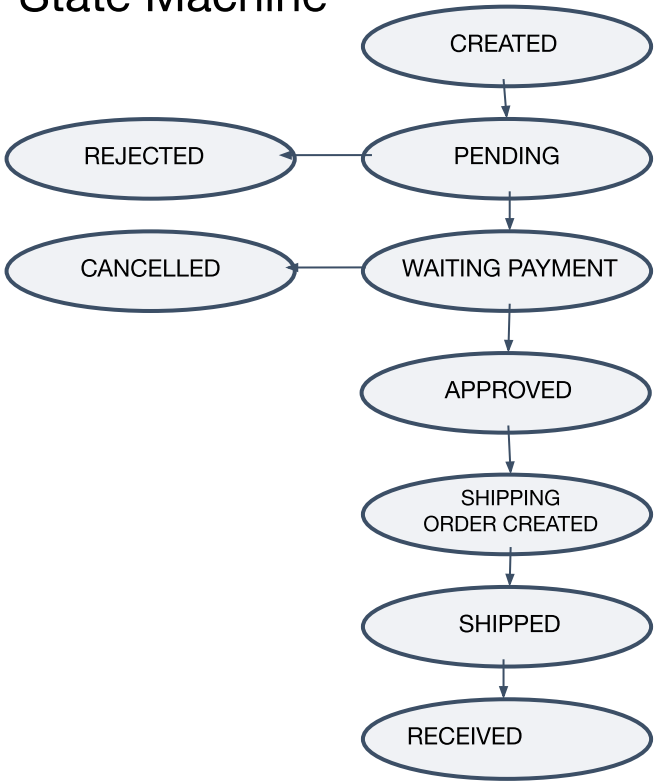
Pedidos
State Machine



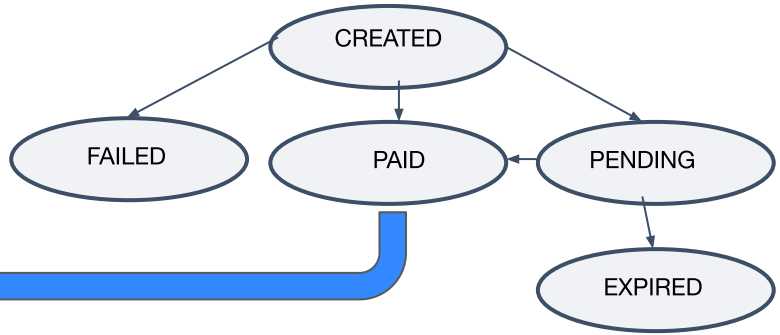
Pagamentos
State Machine



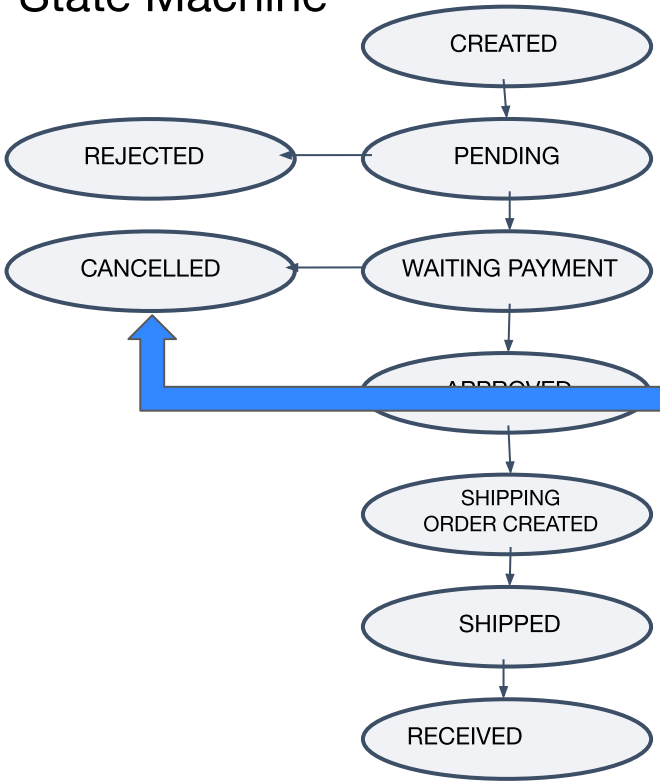
Pedidos
State Machine



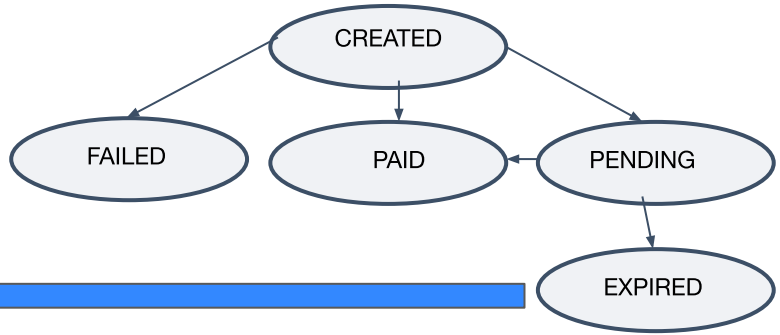
Pagamentos
State Machine



Pedidos
State Machine



Pagamentos
State Machine

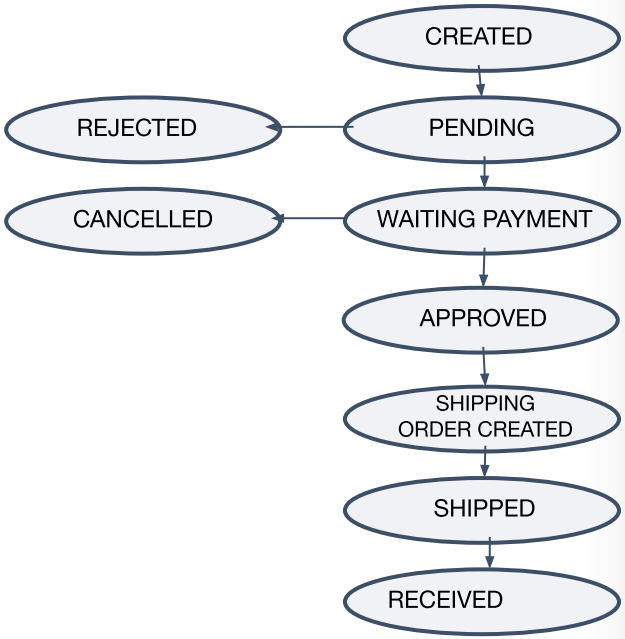


Redesign Hands-on State Machine

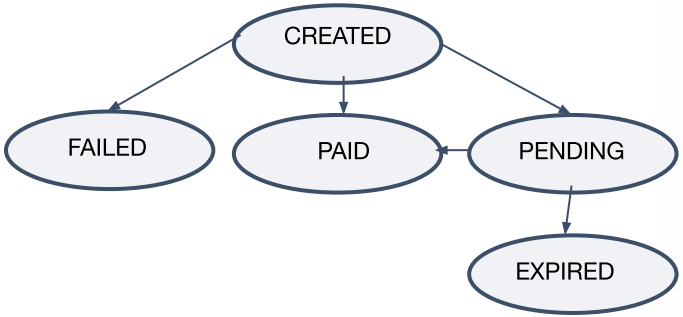
4 Implementar

Não tocar no código antigo

```ruby
PENDING_STATUS = 'PENDING'
APPROVED_STATUS = 'APPROVED'
REJECTED_STATUS = 'REJECTED'
SHIPPING_ORDER_CREATED_STATUS = 'SHIPPING_ORDER_CREATED'
SHIPPED_STATUS = 'SHIPPED'
RECEIVED_STATUS = 'RECEIVED'

POSSIBLE_TRANSITIONS = {
  PENDING: [REJECTED, APPROVED],
  APPROVED: [SHIPPING_ORDER_CREATED],
  SHIPPING_ORDER_CREATED: [SHIPPED],
  SHIPPED: [RECEIVED]
}.freeze

def self.valid_state_transition(last_status, new_status)
  return POSSIBLE_TRANSITIONS[last_status].include?(new_status)
end
```

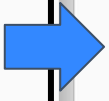



```ruby
class RequestStateMachine
  include Statesman::Machine

  state :created, initial: true
  state :pending
  state :rejected
  state :waiting_payment
  state :cancelled
  state :approved
  state :shipping_order_created
  state :shipped
  state :received

  transition from: :created, to: [:pending]
  transition from: :pending, to: [:rejected, :waiting_payment]
  transition from: :waiting_payment, to: [:approved, :cancelled]
  transition from: :approved, to: [:shipping_order_created]
  transition from: :shipping_order_created, to: [:shipped]
  transition from: :shipped, to: [:received]
end
```

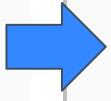



```ruby
class CheckoutStateMachine
  include Statesman::Machine

  state :created, initial: true
  state :pending
  state :failed
  state :paid
  state :expired

  transition from: :created, to: [:pending, :paid, :failed]
  transition from: :pending, to: [:paid, :expired]

  after_transition(to: :pending) do |checkout|
    checkout.wait_payment
  end

  after_transition(to: :paid) do |checkout|
    checkout.approve_request
  end

  after_transition(to: :expired) do |checkout|
    checkout.cancel_request
  end
end
```

```ruby
class CheckoutStateMachine
  include Statesman::Machine

  state :created, initial: true
  state :pending
  state :failed
  state :paid
  state :expired

  transition from: :created, to: [:pending, :paid, :failed]
  transition from: :pending, to: [:paid, :expired]

  after_transition(to: :pending) do |checkout|
    checkout.wait_payment
  end

  after_transition(to: :paid) do |checkout|
    checkout.approve_request
  end

  after_transition(to: :expired) do |checkout|
    checkout.cancel_request
  end
end
```



```ruby
class Checkout
  def wait_payment
    request.wait_payment
  end

  def approve_request
    request.approve
  end

  def cancel_request
    request.cancel
  end
end
```



```ruby
class Request
  def wait_payment
    state_machine.transition_to(:waiting_payment)
  end

  def approve
    state_machine.transition_to(:approved)
  end

  def cancel
    state_machine.transition_to(:cancelled)
  end
end
```

Resolver casos antigos

176 pedidos criados com a solução antiga

status	# pedidos
PENDING	4
SHIPPING_ORDER_CREATED	5
CANCELLED	18
PAYMENT_DECLINED	9
REJECTED	80
SHIPPED	23
RECEIVED	37

Resolver casos antigos **relevantes**

32 pedidos criados com a solução antiga

status	# pedidos
PENDING	**4**
SHIPPING_ORDER_CREATED	**5**
~~CANCELLED:~~	~~18~~
~~PAYMENT_DECLINED~~	~~9~~
~~REJECTED~~	~~80~~
SHIPPED	**23**
~~RECEIVED~~	~~37~~

Resolver casos antigos **relevantes**

32 pedidos criados com a solução antiga

- Worker

  Processar cada pedido antigo com o novo código

- Get state

  Sobrescrever método que retorna o estado

```
request.status
```

```
def status
  self.state_transitions.most_recent.to_state
end
```

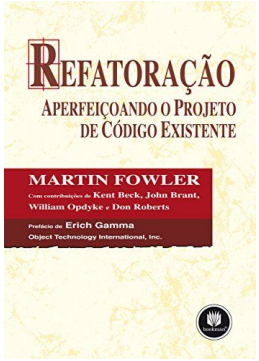
Excluir código antigo

```
PENDING_STATUS = 'PENDING'
APPROVED_STATUS = 'APPROVED'
REJECTED_STATUS = 'REJECTED'
SHIPPING_ORDER_CREATED_STATUS = 'SHIPPING_ORDER_CREATED'
SHIPPED_STATUS = 'SHIPPED'
RECEIVED_STATUS = 'RECEIVED'

POSSIBLE_TRANSITIONS = {
  PENDING: [REJECTED, APPROVED],
  APPROVED: [SHIPPING_ORDER_CREATED],
  SHIPPING_ORDER_CREATED: [SHIPPED],
  SHIPPED: [RECEIVED]
}.freeze

def self.valid_state_transition(last_status, new_status)
  return POSSIBLE_TRANSITIONS[last_status].include?(new_status)
end
```

"Escreva código como se o próximo desenvolvedor a mantê-lo fosse um maníaco homicida que sabe onde você mora."

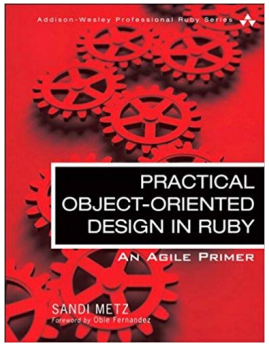
Kathy Sierra


KEEP
CALM
AND
HAVE
FUN



Refactoring: Improving the Design of Existing Code

Martin Fowler



Practical Object-Oriented Design in Ruby: An Agile Primer

Sandi Metz

OBRIGADA :)

PERGUNTAS

Milena Mayumi

Software Engineer @ SumUp

linkedin.com/in/milena-mayumi-costa

milena.mayumi@sumup.com

milenamayumi