
ORACLE
Academy

Princípios Básicos de Java

3–6

Definindo Métodos

```
private void catchFly() {
    if (isTouching(Fly.class)) {
        removeTouching(Fly.class);
    }
}
```

ORACLE
Academy

Objetivos

- Esta lição abrange os seguintes objetivos:
  - Descrever o posicionamento efetivo dos métodos em uma superclasse ou em uma subclasse
  - Simplificar a programação criando e chamando métodos definidos
  - Tratar colisões

ORACLE Academy

Posicionamento Eficiente de Métodos

- Às vezes, são necessárias muitas linhas de código para programar um comportamento
  - Por exemplo, talvez você queira programar uma instância para comer outros objetos ou para virar quando atingir a extremidade do mundo
- Você pode definir novos métodos para simplificar a lógica, economizar tempo e usar menos linhas de código
  - Defina (codifique) um novo método para uma ação dentro do método act
  - Chame (use) o novo método no método act de um mundo ou ator ou dentro de outro método
  - Defina (codifique) o método na superclasse se quiser que as respectivas subclasses herdem automaticamente o método

O uso de métodos permite aproveitar os recursos oferecidos pela equipe de desenvolvimento do Greenfoot.

A definição de nossos próprios métodos permite expandir a funcionalidade dos nossos objetos. Os métodos definidos por você podem ser usados da mesma forma que os métodos herdados fornecidos pelo Greenfoot.

Métodos Definidos

- Os métodos definidos são novos métodos criados pelo programador
- Esses métodos:
  - Podem ser executados imediatamente ou armazenados e chamados mais adiante
  - Não mudam o comportamento da classe quando armazenados
  - Dividem o código em métodos menores, facilitando a leitura

Os métodos definidos criam um novo método que uma classe ainda não tinha. Esses métodos são gravados em um código-fonte da classe dentro do método act.

Embora a definição de seus próprios métodos talvez não melhore o desempenho do jogo, pode proporcionar muitas outras vantagens, como uma melhor legibilidade do código e menor tempo de desenvolvimento.

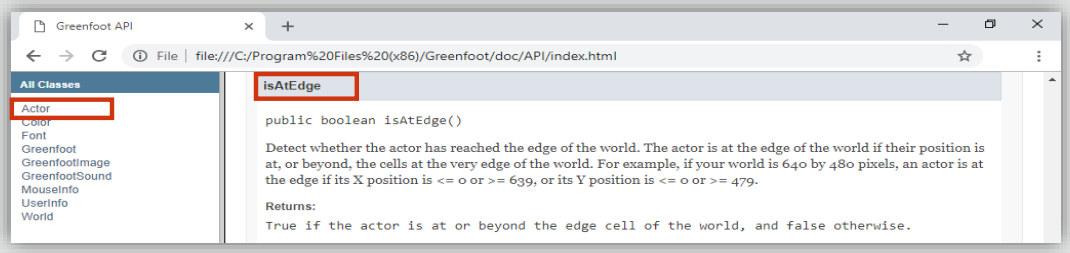
Virar na Extremidade do Mundo

- Problema:
  - As instâncias param e não conseguem mover-se quando atingem a extremidade do mundo
  - Elas devem virar e se mover na direção oposta quando atingirem a extremidade do mundo
- Solução:
  - Crie uma subclasse de Ator que defina um método capaz de detectar se o objeto está na extremidade do mundo e virar de forma apropriada
  - Crie subclasses da subclasse Ator que deve herdar o método
  - Chame (use) os novos métodos nas subclasses que devem virar e se mover na extremidade do mundo

Nem sempre é óbvio quando devemos criar uma subclasse da nossa subclasse. O planejamento do jogo ajuda a identificar esses padrões com antecedência.

Testar se o Ator está na extremidade do mundo

- O Greenfoot tem um método na classe Ator denominado isAtEdge()
- que retornará verdadeiro se o Ator estiver em uma das extremidades
- Podemos usá-lo para detectar uma extremidade e fazer com que os atores virem, em vez de ficarem pairando em uma das extremidades



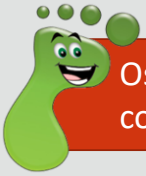
Testar a Posição de um Objeto no Mundo

- Para testar se um objeto está perto da extremidade do mundo, faça o seguinte:
  - Use uma instrução if com o método booliano isAtEdge() para testar se a condição é verdadeira ou falsa
  - Exemplo: podemos girar uma instância em 180 graus se ela estiver na extremidade do mundo

```
public void act()
{
    move(1);
    if(isAtEdge())
    {
        turn(180);
    }
}
```


Girar uma imagem em 180 graus significa que ela girará exatamente a metade de uma rotação completa, ou seja, virará na direção oposta.

Operadores Lógicos



Os operadores lógicos podem ser adicionados às instruções if/else para combinar várias expressões boolianas em uma única expressão booliana

Operador Lógico	Significa	Definição
Ponto de Exclamação (!)	NOT	Reverte o valor de uma expressão booliana (se b for verdadeiro, !b será falso. Se b for falso, !b será verdadeiro).
& duplo (&&)	AND	Combina dois valores boolianos e retorna um valor booliano que será verdadeiro se e somente se os dois operandos forem verdadeiros.
Duas linhas (II)	OR	Combina duas variáveis boolianas ou expressões e retorna um resultado que é verdadeiro se um ou os dois operandos forem verdadeiros.

Para usar a lógica booliana, é comum usar variáveis para testar uma condição. Essas são variáveis que a instância do objeto não armazena e estão associadas ao nível do método. Então, quando um método for terminado, o valor da variável local será perdido. É importante atribuir um nome apropriado a essas variáveis para facilitar a leitura das suas expressões boolianas.

Exemplo de Operadores Lógicos

- Se uma Aranha estiver na metade superior direita do cenário, mova-a para frente:

```
public void act()
{
    if((getX() > getWorld().getWidth()/2) && (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

- Se a posição horizontal do Ator for maior que o centro do mundo horizontal, e a plano vertical do Ator for menor que o centro do mundo vertical…
  - getX() retornará a posição horizontal do Ator
  - getWorld().getWidth() retornará o tamanho horizontal do Mundo
  - getY() retornará a posição vertical do Ator
  - getWorld().getHeight() retornará o tamanho vertical do Mundo

Exemplo de Operadores Lógicos

```
public void act()
{
    if((getX() > getWorld().getWidth()/2) && (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

- Substitua || por && para testar se uma Aranha está na metade superior OU na metade da direita do cenário

```
public void act()
{
    if((getX() > getWorld().getWidth()/2) || (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

Exemplo de Operadores Lógicos

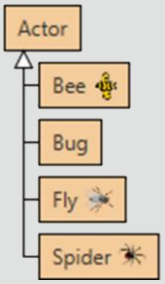
```
public void act()
{
    if((getX() > getWorld().getWidth()/2) || (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

- Use ! para mover uma Aranha para frente se ela NÃO estiver na metade da direita do cenário

```
public void act()
{
    if(!(getX() > getWorld().getWidth()/2) || (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

Criando uma Superclasse

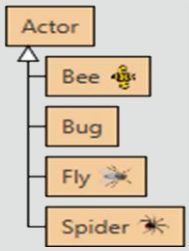
- Para que tanto a classe Aranha quanto a classe Mosca herdem o comportamento de virar, posicionamos o código em uma superclasse
- O nome dessa superclasse deve ser descritivo das classes em comum que herdarão dela
- Criaremos uma classe Inseto que não tem uma imagem e não terá instâncias que atuem no cenário, mas que conterá alguns métodos definidos que outras subclasses herdarão
  - Clique com o botão direito no Ator e escolha New subclass
  - Dê à nova classe o nome de Inseto
  - Não atribua uma imagem
  - Clique em Ok



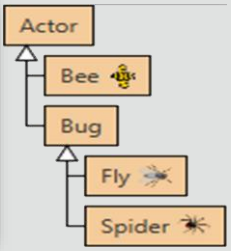
O diagrama mostra que todas as nossas classes são subclasses diretas de Ator.

Criar as Subclasses Besouro

- É possível recriar a Aranha e a Mosca clicando com o botão direito do mouse no Besouro e selecionando a nova Subclasse
- No entanto, como criamos a Aranha e a Mosca anteriormente, podemos modificar o código-fonte delas para estender do Besouro, em vez de estendê-lo de Ator




```
public class Spider extends Bug
{
    /**
     * Act - do whatever the Spider wants to do. Thi
     * the 'Act' or 'Run' button gets pressed in the
     */

    public void act()
    {
        if(!(getX() > getWorld().getWidth()/2) || (g
        {
```



ORACLE Academy

JF 3-6
Definindo Métodos

14

A classe está definida como:
public class Spider extends Actor

Mude para:

public class Spider extends Bug

Isso muda a superclasse Aranha e produz um bom resultado, porque continuamos no mesmo caminho de herança.

Definir o Método turnAtEdge() na Superclasse

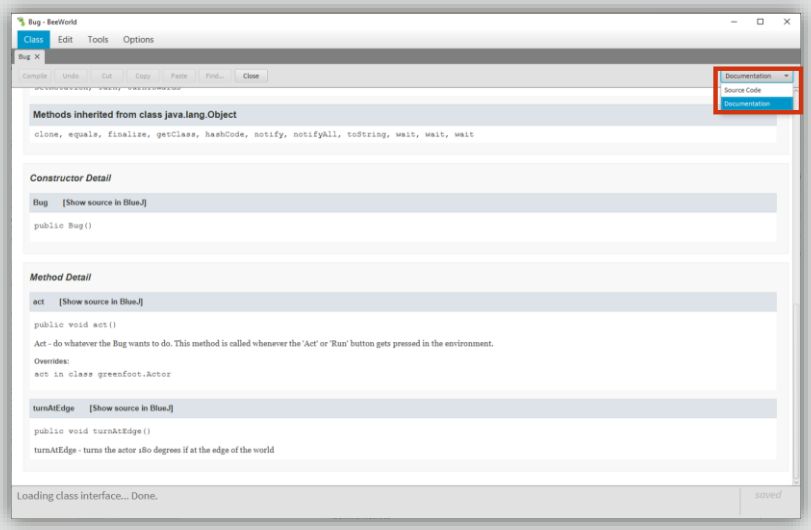
- Abra o Code editor da classe Inseto
- Crie o código do método turnAtEdge() abaixo do método act()

```
public class Bug extends Actor
{
    /**
     * Act - do whatever the Bug wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }

    /**
     * turnAtEdge - turns the actor 180 degrees if at the edge of the world
     */
    public void turnAtEdge()
    {
        if(isAtEdge())
        {
            turn(180);
        }//endif
    }
}
```


Ele foi definido como público porque queremos que as subclasses usem esse método.
Ele é nulo porque não retorna um valor.

Documentação da Classe

- A documentação da classe Besouro mostra o novo método depois de ser definido



ORACLE Academy

JF 3-6
Definindo Métodos

Altere a exibição no editor para mostrar a documentação da caixa suspensa na parte superior direita da janela.

Chamar o Método turnAtEdge() na Subclasse

- Abra o Code editor da subclasse Mosca da lição anterior
- Adicione uma chamada ao método turnAtEdge() dentro do método act()
- Repita isso com a subclasse Aranha

```
public void act()
{
    move(1);
    if (Greenfoot.getRandomNumber(100) < 10)
    {
        turn(Greenfoot.getRandomNumber(90)-45);
    }
    turnAtEdge();
}
```


Observe que podemos chamar (usar) um método de uma superclasse (dependendo de sua visibilidade/acessibilidade), caso ele seja público de uma superclasse.

Definir os Métodos at[left, right, top, bottom]Edge() na classe Abelha

- Na classe Abelha, criaremos métodos que
  - determinarão a extremidade da qual a Abelha está se aproximando
  - moverão a Abelha para a extremidade oposta (parecerá dar a volta)
- Por exemplo, se a Abelha estiver indo para a direita e se aproximar da extremidade direita:
  - desaparecerá da extremidade direita
  - reaparecerá na extremidade esquerda
  - continuará indo em direção à extremidade direita novamente

- Para saber qual extremidade o Ator está tocando, vamos definir quatro métodos separados, um para cada lado

Definir o Método atRightEdge() na classe Abelha

- Abra o Code editor da classe Abelha
- Crie o código do método atRightEdge() abaixo do método act

```
/**
 * Test if we are close to the right edge of the world
 * Return true if the object is.
 */
private boolean atRightEdge()
{
    if(getX() > getWorld().getWidth() - 20)
       return true;
    else
       return false;
}//end method atRightEdge
```


getWorld() retorna uma referência ao mundo atual.
getWorld().getWidth() retorna a largura do mundo atual.
Poderíamos ter usado aqui um valor como 800, mas, se alterássemos a largura do mundo, a funcionalidade desse método seria quebrada. Chamar o método getWidth() do mundo produz uma solução flexível.

Definir o Método atBottomEdge() na classe Abelha

- Abra o Code editor da classe Abelha
- Crie o código do método atBottomEdge() abaixo do método act

```
}//end method act

/**
 * Test if we are close to the bottom edge of the world
 * Return true if the object is.
 */
private boolean atBottomEdge()
{
    if(getY() > getWorld().getHeight() - 20)
       return true;
    else
       return false;
}//end method atBottomEdge
```

Métodos chamados de atRightEdge() e atBottomEdge()

- Os métodos usados em atRightEdge() incluem:
  - getX():
    - um método Ator que retorna a coordenada x da localização atual do ator
  - getY():
    - um método Ator que retorna a coordenada y da localização atual do ator
  - getWorld():
    - um método Ator que retorna o mundo em que o ator vive
  - getHeight():
    - um método da classe Mundo que retorna a altura do mundo
  - getWidth():
    - um método da classe Mundo que retorna a largura do mundo

Sempre é melhor usar os métodos getHeight() e getWidth() do que digitar um valor.

Chamar Métodos na Classe Abelha

- Abra o Code editor da classe Abelha
- Crie uma instrução IF que chame os métodos atRightEdge() e atBottomEdge() como uma condição em act
- Se a Abelha estiver no lado esquerdo, reaparecerá no lado direito, e vice-versa

O movimento sempre será executado

Se a Abelha estiver próxima da extremidade direita, a coordenada x será definida como 6. A posição Y atual é mantida usando o método getY().

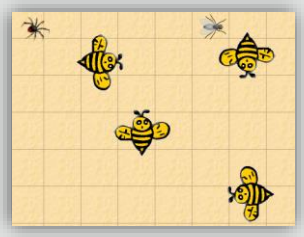
Se a Abelha estiver próxima da extremidade inferior, a coordenada Y será definida como 6. A posição X atual é mantida usando o método getX().

```
public void act()
{
    move(1);
    if(atRightEdge()){
        setLocation(6, getY());
    }else
        if(atBottomEdge()){
            setLocation(getX(), 6);
        }//endif
    //endif
}//end method act
```


Usamos 6 como parte do novo local para que não sejamos detectados imediatamente como estando na extremidade de novo.

Chamar Métodos na Classe

- Abra o Code editor da classe myWorld
- Se necessário,
  - adicione quatro objetos Abelha ao seu mundo
  - selecione Controls -> Save the World



- No método prepare(), adicione o seguinte:
  - bee2.turn(90);
  - bee3.turn(-90);
  - bee4.turn(180);

```
private void prepare()
{
    Fly fly = new Fly();
    addObject(fly,334,42);
    Spider spider = new Spider();
    addObject(spider,46,48);
    Bee bee = new Bee();
    addObject(bee, 150, 100);
    Bee bee2 = new Bee();
    addObject(bee2,388,87);
    Bee bee3 = new Bee();
    addObject(bee3,220,214);
    Bee bee4 = new Bee();
    addObject(bee4,396,312);
    bee2.turn(90);
    bee3.turn(-90);
    bee4.turn(180);
```

Chamar um Método na Classe

- Complete a instrução IF de atLeftEdge() e atTopEdge()
- Teste seu programa para verificar se as instâncias Abelha se distanciam das extremidades do mundo e vão para as extremidades opostas

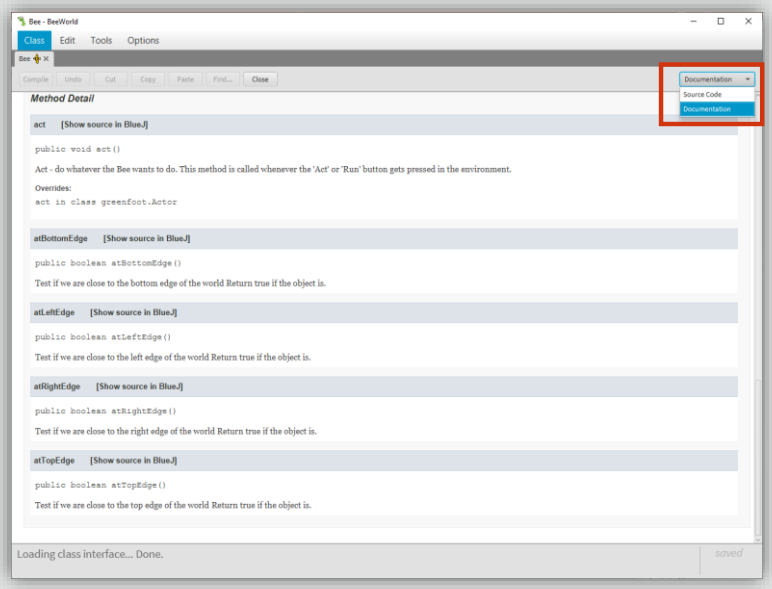
```
}//end method act

/**
 * Test if we are close to the top edge of the world
 * Return true if the object is.
 */
private boolean atTopEdge()
{
    if(getY() < 6)
        return true;
    else
        return false;
}//end method atTopEdge

/**
 * Test if we are close to the left edge of the world
 * Return true if the object is.
 */
private boolean atLeftEdge()
{
    if(getX() < 6)
        return true;
    else
        return false;
}//end method atLeftEdge
```

```
/**
 * Act - do whatever the Bee wants to do. This method is called whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move(1);
    if(atRightEdge()){
        setLocation(6, getY());
    }
    else if(atBottomEdge()){
            setLocation(getX(), 6);
    }
    else if(atLeftEdge()){
            setLocation(getWorld().getWidth()-20, getY());
    }
    else if(atTopEdge()){
            setLocation(getX(), getWorld().getHeight()-20);
    }//endif
}//end method act
```


Agora nossa abelha deverá voar afastando-se de uma extremidade e reaparecendo na extremidade oposta.

Documentação da Classe

- A documentação da classe Abelha mostra o novo método depois de ser definido

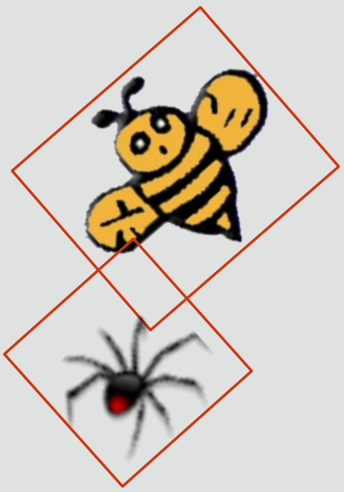


ORACLE Academy

Lembre-se de alterar a exibição no editor. Para mostrar a documentação, alteramos a caixa suspensa na parte superior da janela.

Colisões

- A maioria dos projetos de jogos precisa detectar quando dois atores tocam um no outro, o que costuma ser chamado de colisão
- No Greenfoot, existem várias maneiras de detectar uma colisão
- Estas são algumas delas:
  - `isTouching()`
  - `getOneIntersectingObject(Class)`
  - `getOneObjectAtOffset(Class)`
  - `getIntersectingObjects(Class)`
  - `getNeighbours(distance,diagonal)`
  - `getObjectsAtOffset(dx,dy,Class)`



ORACLE Academy

Apesar de considerarmos uma colisão quando dois atores tocam um no outro, também podemos modificar nosso código para detectarmos uma colisão se dois atores estiverem muito próximos um do outro.
Os últimos dois métodos listados acima retornam uma lista. Precisaríamos, então, processar essa lista para decifrar o que desejamos.

Colisões

Método	Quando Usar
isTouching()	Quando queremos detectar uma colisão com um objeto
getOneIntersectingObject()	Quando desejamos retornar uma referência ao objeto com que ocorreu a colisão. Use para realizar uma ação no objeto com que ocorreu a colisão.
getOneObjectAtOffset()	O mesmo que getOneIntersectingObject(), exceto pelo fato de que é possível alterar o local em que uma colisão será detectada em relação ao objeto atual. Nesse caso, é possível detectar a colisão antes de ela acontecer para, por exemplo, evitar que um ator bata em uma parede.

isTouching() retorna um valor booliano (verdadeiro ou falso). Os outros dois métodos retornam uma referência a um ator.

Método Definido para Remover Objetos

- Você pode escrever o código no seu jogo para que um objeto predador possa comer objetos presa
- Crie um método definido na classe Abelha chamado catchfly() para remover as moscas que a Abelha captura
- Para criar esse método definido, vamos usar a detecção de colisão mais simples: isTouching()
- Este método detecta uma colisão de Abelha com uma Mosca e a remove

```
private void catchFly(){
      if(isTouching(Fly.class)){
         removeTouching(Fly.class);
       }//endif
}
```

ORACLE Academy

O código para isso pode ser criado de várias maneiras.

Se tivéssemos usado apenas isTouching() (com os parênteses vazios), esse método retornaria verdadeiro se tocássemos em qualquer outro ator, inclusive em uma instância de outra Abelha.

Definir o método catchfly() — Alternativa

- Como alternativa, poderíamos ter usado getOneIntersectingObject() e acessado uma referência ao ator antes de excluí-lo

```
}//end method act

/**
 * catchFly2 - if the Bee touches a fly the fly is removed
 */
private void catchFly2(){
    Actor fly = getOneIntersectingObject(Fly.class);
    if(fly != null){
       getWorld().removeObject(fly);
    }//endif
}

/**
 * catchFly - if the Bee touches a fly the fly is removed
```


getOneIntersectingObject() retorna uma referência a um ator. O tipo de ator neste exemplo está restrito a apenas uma instância da classe Mosca. Se ele não encontrar uma colisão com uma Mosca, a variável de referência Mosca receberá o valor nulo.

Vamos testar, então, se a mosca é nula e, caso não seja, vamos removê-la.

Chamar catchfly() no Método act()

- Chame o novo método catchfly()no método act() da Abelha
- Adicione o código fora da instrução if, pois queremos que a abelha sempre tente capturar uma mosca
- Execute o cenário para testar o código

```
public void act()
{
    move(1);
    if(atRightEdge()){
        setLocation(6, getY());
    }
    else if(atBottomEdge()){
            setLocation(getX(), 6);
    }
    else if(atLeftEdge()){
            setLocation(getWorld().getWidth()-20, getY());
    }
    else if(atTopEdge()){
            setLocation(getX(), getWorld().getHeight()-20);
    }//endif
    catchFly();
}//end method act
```

Chamar catchfly() no Método act()

- Mudanças adicionais recomendadas para o método act():
  - reposicione a instrução move() do método act() em um novo método, chamado handleMovement()
  - reposicione as instruções if do método act() que verificam se a Abelha está na extremidade adicionando um método turnAtEdge() que testa se a Abelha está na extremidade

```
public void act()
{
    handleMovement();
    turnAtEdge();
    catchFly();
}//end method act
```

```
private void handleMovement(){
    move(1);
}//end method handleMovement
```

```
private void turnAtEdge(){
    if(atRightEdge()){
        setLocation(6, getY());
    }
    else if(atBottomEdge()){
            setLocation(getX(), 6);
    }
    else if(atLeftEdge()){
            setLocation(getWorld().getWidth()-20, getY());
    }
    else if(atTopEdge()){
            setLocation(getX(), getWorld().getHeight()-20);
    }//endif
}//end method turnAtEdge
```


A prática recomendada é criar métodos para ações de Ator e chamá-los no método act().

Terminologia

- Estes são os principais termos usados nesta lição:
  - Métodos definidos
  - Colisões

Resumo

- Nesta lição, você aprendeu os seguintes tópicos:
  - Descrever o posicionamento efetivo dos métodos em uma superclasse ou em uma subclasse
  - Simplificar a programação criando e chamando métodos definidos

ORACLE Academy


ORACLE
Academy