
ORACLE
Academy

Java Fundamentals

3-9

Abstração

```
public class Spider extends Bug
{
    public void act()
    {
        turnAtEdge();
        move(1);
        BeeWorld myworld = (BeeWorld)getWorld();
        Bee bee = myworld.getBee();
        this.turnTowards(bee.getX(), bee.getY());
    }
}
```

ORACLE
Academy

Objetivo

- Esta lição abrange os seguintes objetivos:
  - Definir abstração e fornecer um exemplo de quando ela é usada
  - Definir a conversão

Abstração

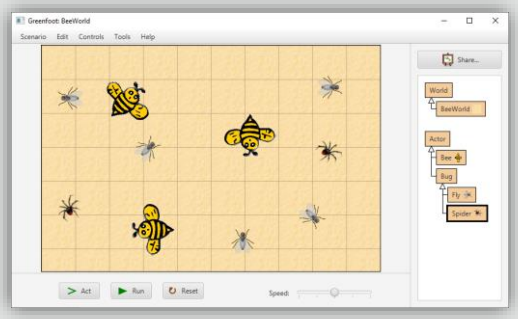
- É possível programar uma nova instância para executar uma única tarefa específica, como reproduzir um som quando uma tecla específica do teclado é pressionada ou exibir um conjunto de perguntas e respostas cada vez que um jogo é iniciado
- Para criar programas em uma escala maior, por exemplo, um programa que crie 10 objetos que executem ações diferentes cada um, é necessário elaborar instruções de programação que permitam criar objetos que executem tarefas diferentes apenas fornecendo os detalhes das diferenças

É possível definir a abstração de muitas formas. Vamos nos concentrar na ideia nos afastando de uma função específica e indo para uma mais geral.

Pense na abstração como o processo de passar de uma tarefa específica para uma mais geral. Então, em vez de chamar um construtor que sempre executa as mesmas tarefas, podemos transmitir valores que nos permitam alterar a configuração inicial.

Exemplo de Abstração

- Por exemplo, se você pretende criar 10 objetos de modo programático, todos posicionados em locais diferentes, não é uma estratégia eficiente elaborar 10 linhas de código para cada objeto
- Em vez disso, você abstrai o código e elabora instruções mais genéricas para lidar com a criação e o posicionamento dos objetos



Já começamos a dominar a abstração criando nossos próprios métodos e construtores.

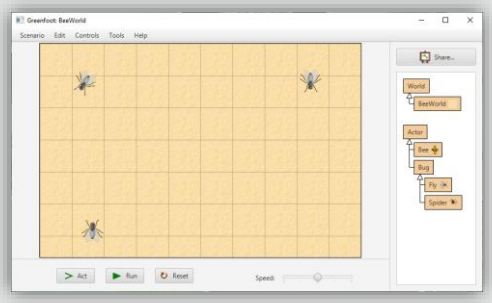
Princípio da Abstração

- O uso da abstração tem como objetivo reduzir a duplicação de informações em um programa
- O princípio da abstração pode ser um pensamento geral, como "não se repita"
- Por exemplo, você deseja criar um tabuleiro de jogo que tenha blocos, árvores, bastões e widgets:
  - Não é necessário escrever instruções de programação repetitivas para adicionar cada um desses itens
  - Em vez disso, você pode abstrair o procedimento para simplesmente adicionar objetos a um tabuleiro de jogo em um local específico

Ao elaborar o código-fonte e descobrir que algo que você esteja escrevendo é muito semelhante a outro código que você já escreveu, verifique se consegue abstrair sua finalidade para outro método. Em seguida, chame isso dos locais corretos. Será produzido um código muito mais fácil de manter.

Exemplo de Pseudocódigo de Abstração

- Por exemplo, quando você adiciona uma mosca ao Mundo, ela também terá uma velocidade máxima na qual poderá se mover e uma direção inicial
- Seu código adicionará uma mosca e especificará a velocidade máxima na qual ele poderá se mover e a direção inicial
- Veja a seguir o pseudocódigo:
  - Create new Fly (4,90)
  - Create new Fly (2.120)
  - Create new Fly (3.270)



ORACLE Academy

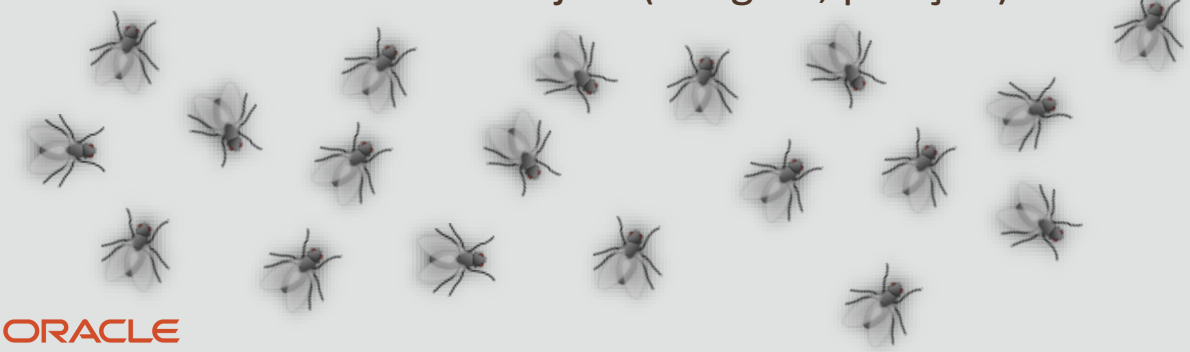
Poderíamos ter escrito

```
Fly fly1 = new Fly();
fly1.setSpeed(4);  // a defined method
fly1.setRotation(90);
```

Teríamos que repetir o código acima para cada mosca. Com a abstração de sua finalidade para o construtor, ele reduz grande parte da repetição.

Exemplo de Pseudocódigo de Abstração

- Imagine o código necessário para 300 imagens de moscas
- Para implementar a abstração, crie um método que gere um novo objeto posicionado onde for necessário e exiba a imagem apropriada
- Chame o método: newObject (imagem, posição)



ORACLE Academy

A equipe de programação Greenfoot criou métodos para facilitar o desenvolvimento. Você pode abstraí-lo ainda mais para torná-lo mais poderoso.

Técnicas de Abstração

- A abstração ocorre de muitas maneiras na programação
  - Uma técnica é abstrair o código de programação usando variáveis e parâmetros para transmitir tipos diferentes de informações a uma instrução
  - Outra técnica é identificar instruções de programação semelhantes em diferentes partes de um programa que podem ser implementadas em apenas um local abstraindo as partes variáveis

Você, quase sempre, modificará os construtores para que possa transmitir informações iniciais a eles.

Exemplo de Técnicas de Abstração

- Por exemplo, em um jogo onde você pega outros objetos, é possível aumentar o placar com um valor diferente, dependendo do objeto que foi pego

+ 1

- 1

- Você pode usar a abstração fazendo com que um evento aumente o placar por meio de um parâmetro em vez de um valor definido


```
public void increaseScore(int value) {
  score = score + value;
}
```

É possível chamar isso de uma área como increaseScore(5) e de outra área como increaseScore(10).

Construtor Usando Variáveis

- Neste exemplo, a mosca tem uma variável definida para armazenar a velocidade atual
- O construtor, aleatoriamente, gera um número até a velocidade máxima transmitida pelo parâmetro
- Também configuramos a rotação inicial da mosca

```
public class Fly extends Bug
{
    private int speed;

    /**
     * Fly - sets the initial values of the fly
     */
    public Fly(int maxSpeed, int direction){
        speed = (Greenfoot.getRandomNumber(maxSpeed)+1);
        setRotation(direction);
    }//end constructor Fly(int, int)
```


Adicionamos 1 à velocidade porque getRandomNumber() pode retornar um 0. Uma mosca com uma velocidade de zero é fácil de pegar!

Construtor Usando Variáveis

- O método randomMovement() que moveu a mosca a uma velocidade constante de um com move(1) é atualizado para usar a velocidade da variável da instância
  - move(speed)

```
public void act()
{
    randomMovement();
    turnAtEdge();
}

private void randomMovement(){
    move(speed);
    if (Greenfoot.getRandomNumber(100) < 10)
    {
        turn(Greenfoot.getRandomNumber(90)-45);
    }//endif
}//end method randomMovement
```


Se quiser que a velocidade, rotação, etc. seja alterada durante um jogo, crie uma variável de classe para armazenar o valor atual.

Programando para Posicionar Instâncias

- Depois de definir as variáveis de velocidade e direção em um construtor, escreva i código de programação para adicionar automaticamente instâncias da classe ao mundo
- A instrução de programação a seguir adicionada à classe BeeWorld:
  - Cria uma nova instância da mosca sempre que o BeeWorld é reinicializado, com uma velocidade e uma direção específicas
  - Coloca a instância no BeeWorld nas coordenadas de x e y específicas

```
addObject (new Fly(2, 90), 150, 150);
```


Isso incluirá uma mosca nas coordenadas (150,150) com uma velocidade máxima aleatória de 2 começando em uma direção de 90 graus.

Exemplo de Construtor

- Examine as instruções addObject() no construtor BeeWorld ao adicionar uma nova mosca

```
/**
 * Constructor for objects of class BeeWorld.
 *
 */
public BeeWorld()
{
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
    super(600, 400, 1);
    addObject (new Bee(), 150, 100);
    addObject(new Spider(), 510, 360);

    addObject (new Fly(4, 90), 505, 70);
    addObject (new Fly(2, 120), 83, 73);
    addObject (new Fly(3, 270), 98, 350);

}
```


Agora estamos adicionando quatro moscas cada com uma velocidade aleatória máxima, uma direção inicial e um local inicial diferente. Tudo com apenas quatro linhas de código no construtor BeeWorld.

Abstrair Código para um Método

- Você pode prever a abstração durante a fase de design de um projeto ou examinar o código de programação para identificar instruções que podem se beneficiar da abstração
- Você reconhecerá, com muita frequência, oportunidades de abstrair instruções de programação ao escrever linhas de código que parecem repetitivas

Escrever linhas que você já escreveu antes, geralmente, é o gatilho para o início da abstração.

Exemplo de Abstração de Código para um Método

- Examine o código abaixo e no slide a seguir

```
public BeeWorld()
{
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
    super(600, 400, 1);
    prepare();
}

/**
 * Prepare the world for the start of the program.
 * That is: create the initial objects and add them to the world.
 */
private void prepare()
{
    addObject (new Bee(), 150, 100);
    addObject(new Spider(), 510, 360);

    addObject (new Fly(4, 90), 505, 70);
    addObject (new Fly(2, 120), 83, 73);
    addObject (new Fly(3, 270), 98, 350);
    addObject (new Fly(2, 190), 150, 10);
    addObject (new Fly(3, 120), 130, 20);
    addObject (new Fly(1, 270), 5, 10);
    addObject (new Fly(1, 90), 200, 10);
    addObject (new Fly(2, 120), 300, 110);
    addObject (new Fly(3, 270), 130, 120);
}//end method prepare
```

Exemplo de Abstração de Código para um Método

```
public BeeWorld()
{
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
    super(600, 400, 1);
    prepare();
}

/**
 * Prepare the world for the start of the program.
 * That is: create the initial objects and add them to the world.
 */
private void prepare()
{
    addObject (new Bee(), 150, 100);
    addObject(new Spider(), 510, 360);

    for(int i=0; i<9; i++){
        int maxSpeed = Greenfoot.getRandomNumber(4)+1;
        int direction = Greenfoot.getRandomNumber(360);
        int xCoord = Greenfoot.getRandomNumber(this.getWidth());
        int yCoord = Greenfoot.getRandomNumber(this.getHeight());

        addObject(new Fly(maxSpeed, direction),xCoord, yCoord);
    }//end for
}//end method prepare
```

ORACLE Academy

JF 3-9
Abstração

17

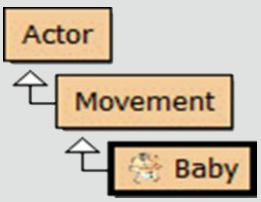
Criamos 10 moscas cada com uma direção e uma posição aleatórias na tela.

Isso é feito usando o loop for. O loop for executará o código inserido em seus colchetes uma quantidade definida de vezes. Nesse exemplo, são 10 loops. Examinaremos os loops mais adiante.

Poderíamos aumentar a velocidade máxima das moscas no decorrer do jogo para tornar mais difícil pegá-las.

Abstração de Código Simples

- E se você tivesse pouco conhecimento do Greenfoot ou de programação e quisesse criar um jogo para mover um bebê pela tela?
- Bastaria simplificar a forma de mover um objeto Ator pela tela criando um conjunto simples de métodos de movimento: moveRight(), moveLeft(), moveUp() e moveDown()
- Isso fornece uma abstração mais simples do que a API do Greenfoot padrão com seus métodos setLocation() e getX()/getY() incorporados



Se você pretende ter atores com funções compartilhadas, é melhor criar uma subclasse de ator, adicionar a funcionalidade e, em seguida, criar uma subclasse da classe. Lembre-se que a classe Movimento pode ter muitas subclasses abaixo dela.

Criar a Subclasse Bebê

- Crie uma subclasse da classe Ator denominada Movimento que permitiria ao jogador mandar o ator Bebê se mover na direção desejada
- Adicione ao Movimento o código a seguir, que definirá a quantidade de movimento toda vez que um movimento for necessário

```
public class Movements extends Actor
{
    //  An actor superclass that provides movement in four directions.
    private static final int SPEED = 4;

    /**
     * Act - do whatever the Movements 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.
    }
}
```


private static final int SPEED= 4

Cria uma constante de classe que podemos usar em nosso código. A vantagem é que ela facilita a leitura de nosso código quando a usamos e também nos permite alterar um valor para aumentar a velocidade de todos os atores que a usam.

Criar Métodos de Movimento para a Subclasse Bebê

- Em seguida, adicione os seguintes métodos de movimento:
  - Esses métodos simplificam e abstraem a API do Greenfoot de getX()/getY()

```
}//end method act

private void moveRight()
{
    setLocation ( getX() + SPEED, getY() );
}//end method moveRight

private void moveLeft()
{
    setLocation ( getX() - SPEED, getY() );
}//end method moveLeft

private void moveUp()
{
    setLocation ( getX(), getY() - SPEED);
}//end method moveUp

private void moveDown()
{
    setLocation ( getX(), getY() + SPEED);
}//end method moveDown
```


setLocation() move um ator sem precisar alterar sua rotação. Em alguns casos, é melhor do que move() e rotate(). No exemplo, queremos que um bebê se mova pela tela. Não queremos que ele gire, mas que fique sempre em pé, mas ainda se mova pela tela em direções diferentes.

Codificar o Método act()

- Codifique o método act() da classe de ator Bebê de forma que suas instâncias se movam quando as teclas de seta forem pressionadas
- Essa abstração oculta e automatiza o código mais complexo, apenas mostrando moveLeft(), moveRight(), etc

```
public void act()
{
    if (Greenfoot.isKeyDown("left") )
    {
        moveLeft();
    }//endif

    if (Greenfoot.isKeyDown("right") )
    {
        moveRight();
    }//endif

    if (Greenfoot.isKeyDown("up") )
    {
        moveUp();
    }//endif

    if (Greenfoot.isKeyDown("down") )
    {
        moveDown();
    }//endif
}//end method act
```


Se o bebê se movesse em uma direção depois de fazer contato com outro ator, também poderíamos chamar esse método nesse local.

```
if (leftBump()) {
  moveLeft();
}
```

Acessando Métodos em Outras Classes

- Às vezes, queremos acessar métodos e propriedades em outras classes
- Durante uma colisão, podemos obter uma referência ao objeto colidido usando um método como getOneIntersectingObject()

Em alguns cenários que você elaborar, nunca precisará se preocupar com ações que acontecem longe do(s) ator(es) principal(is), mas em outros jogos, você precisará monitorá-las constantemente.

Acessando Métodos em Outras Classes

- Também temos a opção de chamar o método Mundo que retornaria todos os objetos e, em seguida, localizar o que queremos
- Mas e se quisermos acessar outro ator ou método fora de uma colisão ou não quisermos iterar em todos os objetos?

Em alguns cenários que você elaborar, nunca precisará se preocupar com ações que acontecem longe do(s) ator(es) principal(is), mas em outros jogos, você precisará monitorá-las constantemente.

Estendendo a Classe BeeWorld

- Poderíamos ter mantido a pontuação na classe BeeWorld
- Em seguida, criado métodos para ler e atualizar a pontuação
- Isso permitira a fácil atualização da pontuação de qualquer ator

```
public class BeeWorld extends World
{
    private int score;

    /**
     * Constructor for objects of class BeeWorld.
     *
     */
    public BeeWorld()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(600, 400, 1);
        prepare();
    }

    public int getScore(){
        return score;
    }//end method getScoe

    public void updateScore(){
        score++;
        showText("Score : " + score, 60, 390);
    }//end method updateScore
```


Observe que mantivemos a propriedade pontuação privada de forma que nenhum ator fora da classe BeeWorld tivesse acesso direto. Eles devem usar os métodos públicos getScore() e updateScore().

Acessando Outros Métodos de Objetos

- Na classe Abelha, poderíamos acessar o método de pontuação usando o método getWorld()
- Você pode tentar:

```
private void updateScore(){
    World myworld = getWorld();
    myworld.updateScore();
}//end method update
```

cannot find symbol - method updateScore()

  - Mas isso produzirá um erro porque o tipo de retorno do getWorld() é Mundo, e ele não contém um método para updateScore()
  - Embora nosso BeeWorld seja um tipo de Mundo, o Java ainda verificará os métodos de Mundo e se o método não estiver presente, será gerado um erro
  - Para este exemplo, usaremos uma conversão

Conversão é um termo com o qual você se tornará muito familiarizado durante este curso.

Conversão

- Conversão é quando queremos dizer ao compilador Java que uma classe que estamos acessado, na verdade, é um tipo mais específico de classe
- Em nosso exemplo anterior, queremos dizer ao compilador que a classe Mundo é, na verdade, uma classe BeeWorld
- Para isso, fazemos a conversão dela
- Então...

```
private void updateScore(){
    World myworld = getWorld();
    myworld.updateScore();
}//end method updateScore
```

- Torna-se...

```
private void updateScore(){
    BeeWorld myworld = (BeeWorld) getWorld();
    myworld.updateScore();
}//end method updateScore
```

ORACLE Academy

Não podemos converter nada para que se torne outra coisa. Deve haver uma relação, como Mundo e BeeWorld ou Ator e Abelha. Não podemos converter um Mundo em um Ator, por exemplo.

Acessando Outros Atores

- É possível acessar outros atores fora das colisões de forma semelhante à que acessamos métodos
- Crie um campo privado e um método público para retorná-lo

```
public class BeeWorld extends World
{
    private int score;
    private Bee bee = new Bee();

    /**
     * Constructor for objects of class BeeWorld.
```

```
    private void prepare()
    {
        addObject (bee, 150, 100);
        addObject(new Spider(), 510, 360);

        for(int i=0; i<9; i++){
            int maxSpeed = Greenfoot.getRandomNumber(4)+1;
            int direction = Greenfoot.getRandomNumber(360);
            int xCoord = Greenfoot.getRandomNumber(this.getWidth());
            int yCoord = Greenfoot.getRandomNumber(this.getHeight());

            addObject(new Fly(maxSpeed, direction),xCoord, yCoord);
        }//end for
    }//end method prepare

    public Bee getBee(){
        return bee;
    }//end method getBee
}
```

ORACLE Academy

Você deve observar que isso é muito semelhante ao nosso último exemplo. Criamos um campo de classe privada para armazenar o tipo de dados que queremos acessar. Em seguida, criamos um método público para retornar esse valor.

Acessando Outros Atores

- Para obter acesso a esse método, faríamos o seguinte:
- Neste exemplo, agora a Aranha se moveria diretamente na direção da Abelha

```
public class Spider extends Bug
{
    /**
     * Act - do whatever the Spider wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */

    public void act()
    {
        turnAtEdge();
        move(1);
        BeeWorld myWorld = (BeeWorld) getWorld();
        Bee bee = myWorld.getBee();
        this.turnTowards(bee.getX(), bee.getY());
    }//end method act
}
```


BeeWorld myworld = (BeeWorld)getWorld();

Essa linha obtém uma referência ao mundo atual e a armazena na variável myworld. Como o tipo de retorno do getWorld é Mundo, temos que convertê-lo em um tipo BeeWorld.

Bee bee = myworld.getBee();

Essa linha cria uma variável de abelha que fará referência à Abelha retornada por nosso método getBee();

Assim que tivermos essa referência, poderemos usar a abelha retornada para acessar métodos da instância Abelha.

Usamos os métodos getX() e getY() da instância Abelha no método de ator turnTowards. Em seguida, isso significará que a Aranha sempre girará para encarar a instância Abelha no mundo.

Terminologia

- Os principais termos usados nesta lição foram:
  - Abstração
  - Conversão

Resumo

- Nesta lição, você deverá ter aprendido a:
  - Definir abstração e fornecer um exemplo de quando ela é usada
  - Definir a conversão

ORACLE Academy


ORACLE
Academy