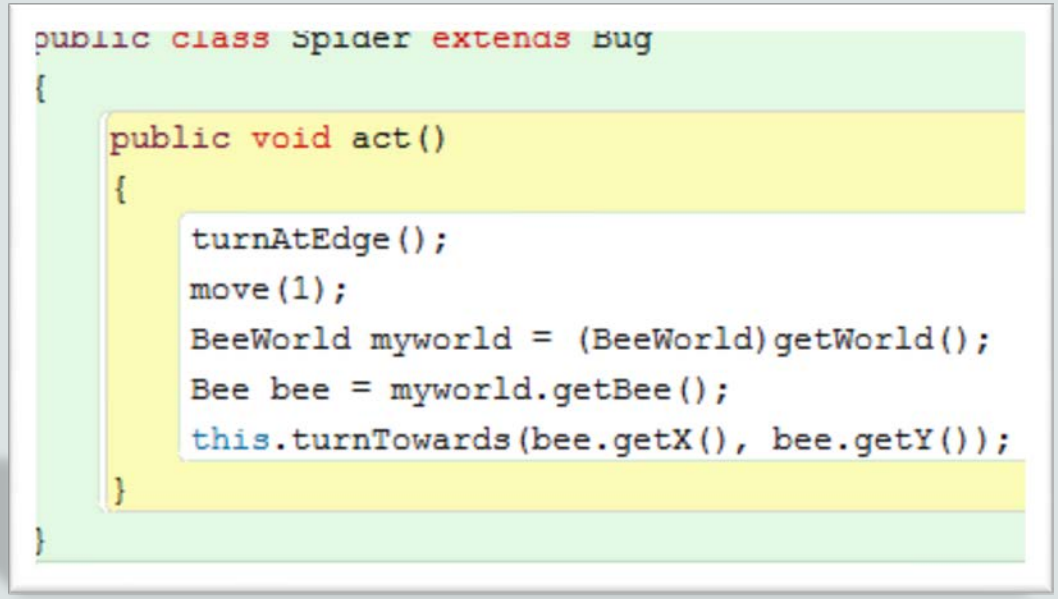

ORACLE
ACADEMY

Java Fundamentals

3-9
Abstraction



ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Define abstraction and provide an example of when it is used
- Define casting

Abstraction

- You can program a new instance to perform a single, specific task, such as play a sound when a specific keyboard key is pressed, or display a set of questions and answers every time a game is started.
- To create programs on a larger scale, for example one that creates 10 objects that each perform different actions, you need to write programming statements that let you repeatedly create objects that perform different tasks, by just providing the specifics for the differences.

Abstraction Example

- For example, if you are going to create 10 objects programmatically, all placed in different locations, it is inefficient to write 10 lines of code for each object.
- Instead, you abstract the code and write more generic statements to handle the creation and positioning of the objects.

Abstraction Principle

- Abstraction aims to reduce duplication of information in a program by making use of abstractions.
- The abstraction principle can be a general thought such as "don't repeat yourself."
- For example, you want to create a game board that has blocks, trees, sticks, and widgets.
  - You do not need to write repetitive programming statements to add each of these items.
  - Instead, you can abstract the procedure to simply add objects to a game board in a specific location.

Abstraction Pseudocode Example

- For example, when you add a Fly to the World it will also have a maximum speed that it can move and an initial direction.
- Your code will add a Fly and specify the maximum speed it can move and the starting direction.
- Here is the pseudo code:
  - Create new Fly (4,90)
  - Create new Fly (2,120)
  - Create new Fly (3,270)

Abstraction Pseudocode Example

- Imagine the code needed for 300 Fly images.
  - To implement abstraction, create a method that creates a new object that is positioned where needed and displays the appropriate image.
    - Call the method: newObject (image, position)

Abstraction Techniques

- Abstraction occurs many ways in programming.
  - One technique is to abstract programming code using variables and parameters to pass different types of information to a statement.
  - Another technique is to identify similar programming statements in different parts of a program that can be implemented in just one place by abstracting out the varying parts.

Abstraction Techniques Example

- For example, in a game where you catch other objects you could increase the score by a different value depending on what object was caught.
- You could use abstraction by having an event increase the score by using a parameter rather than a set value.

Constructor Using Variables

- In this example, the Fly has a variable defined to store the current speed.
- The constructor randomly generates a number up to the maximum speed passed in via the parameter.
- We also setup the initial rotation of the Fly.

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

    public Fly(int maxSpeed,int direction) {
        speed = Greenfoot.getRandomNumber(maxSpeed)+1;
        setRotation(direction);

    }
```

Constructor Using Variables

- The randomMovement() method that moved the fly at a constant speed of one with move(1) is updated to use the instance variable speed.
  - move(speed)

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

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

Programming to Place Instances

- After speed and direction variables are defined in a constructor, write programming code to automatically add instances of the class to the world.
- The following programming statement added to the BeeWorld class:
  - Creates a new instance of Fly each time BeeWorld is re-initialized, with a specific speed and direction.
  - Places the instance in BeeWorld at the specific x and y coordinates.

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

Constructor Example

- Examine the addObject() statements in the BeeWorld constructor when adding a new Fly.

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

    addObject(new Bee(), 310, 201);
    addObject(new Spider(), 510, 360);

    addObject(new Fly(1,90), 505, 70);
    addObject(new Fly(2,180), 83, 73);
    addObject(new Fly(2,270), 98, 352);
    addObject(new Fly(2,110), 505, 350);
}
```

Abstract Code to a Method

- You can anticipate abstraction during the design phase of a project, or you can examine programming code to identify statements that would benefit from abstraction.
- Often times you will recognize an opportunity to abstract programming statements when writing lines of code that appear repetitive.

Abstract Code to a Method Example

- Examine the code below and on the following slide.

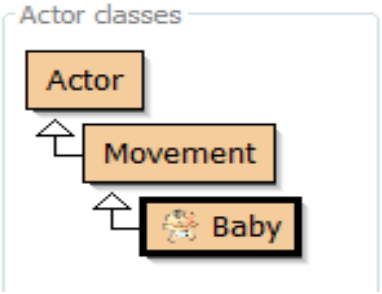
```
import greenfoot.*;

public class BeeWorld extends World
{
    /**
     * Constructor for objects of class BeeWorld.
     *
     */
    public BeeWorld()
    {
        super(600, 400, 1);
        addObject (new Fly(2, 90), 1, 200);
        addObject (new Fly(1, 180), 200, 10);
        addObject (new Fly(3, 270), 300, 50);
        addObject (new Fly(2, 190), 150, 110);
        addObject (new Fly(3, 290), 130, 120);
        addObject (new Fly(2, 50), 5, 10);
        addObject (new Fly(1, 0), 200, 10);
        addObject (new Fly(2, 30), 300, 110);
        addObject (new Fly(2, 190), 150, 110);
        addObject (new Fly(3, 290), 130, 120);
    }
}
```

Abstract Code to a Method Example

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class BeeWorld extends World
{
    int currentMaxSpeed = 2;

public BeeWorld()
    {
        super(600, 400, 1);

        for (int i=0; i<10; i++) {
                int direction = Greenfoot.getRandomNumber(360);
                int xcoord = Greenfoot.getRandomNumber(600);
                int ycoord = Greenfoot.getRandomNumber(400);
                addObject (new Fly(currentMaxSpeed , direction),xcoord,ycoord);
        }
    }
}
```

Simple Abstraction of Code

- What if you had little knowledge of Greenfoot or programming, and wanted to create a game to move a baby around the screen?
  - You could simplify how to move an Actor object around the screen by creating a simple set of movement methods: moveRight(), moveLeft(), moveUp() and moveDown().
  - This provides a simpler abstraction than the standard Greenfoot API with its built-in setLocation() and getX()/getY() methods.



Create Baby Subclass

- Create a subclass of the Actor class called Movement that would allow the player to tell the Baby actor to move in a desired direction.
- Add the following code to Movement which will set the amount of movement each time a move is required.

```
import greenfoot.*;
//  An actor superclass that provides movement in four directions.

public class Movements extends Actor {

          private static final int SPEED= 4;

}
```

Create Movement Methods for Baby Subclass

- Then, add the following movement methods:
  - These methods simplify and abstract the Greenfoot API of getX()/getY().

```
public void moveRight()
{
      setLocation ( getX() + SPEED, getY() );
}
public void moveLeft()
{
      setLocation ( getX() - SPEED, getY() );
}
public void moveUp()
{
      setLocation ( getX(), getY() - SPEED);
}
public void moveDown()
{
      setLocation ( getX(), getY() + SPEED);
}
```

Code the act() Method

- Code the act() method of the Baby Actor class so that its instances move when the arrow keys are pressed.
- This abstraction hides and automates the more complex code, only showing moveLeft(), moveRight(), etc.

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

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

Accessing Methods in Other Classes

- Sometimes we want to access methods and properties in other classes.
- During a collision we can gain a reference to the collided object by using a method like getOneIntersectingObject().
- We also have the option of calling the World method that would return all objects and then locating the one we want.
- But what happens if we want to access another actor or method outside of a collision or we don't want to iterate through all objects?

Extending the BeeWorld Class

- We could have kept the score in the BeeWorld class.
- Then created methods to read and update the score.
- This would then allow easy updating of score from any actor.

```
public class BeeWorld extends World
{
    private int score = 0;

    public int getScore() {
        return score;
    }

    public void updateScore() {
        score++;
        //could update score panel here
    }
```

Accessing Other Objects Methods

- In the Bee class we could access the score method by using the getWorld() method. You might try:

```
World myworld = getWorld();
myworld.updateScore();
```

- But this will produce an error because getWorld() return type is World and World does not contain a method for updateScore().
- Although our BeeWorld is a type of World, Java will still check the methods of World and if the method is not there, an error is produced.
- For this example, we will use a cast.

Casting

- Casting is when we want to tell the Java compiler that a class we are accessing is really another, more specific type of class.
- In our previous example we want to tell the compiler that the World class is really a BeeWorld class.
- To do this we cast it. So…

```
World myworld = getWorld();
myworld.updateScore();
```

- Becomes…

```
BeeWorld myworld = (BeeWorld)getWorld();
myworld.updateScore();
```

Accessing Other Actors

- Accessing other actors outside of collisions can be done in a similar manner to accessing methods.
- Create a private field and a public method to return it.

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

    public Bee getBee() {
        return bee;
    }

    public BeeWorld()
    {
        // Create a new world with 6
        super(600, 400, 1);
        addObject(bee, 298, 176);
```

Accessing Other Actors

- To gain access to this method we would do the following.
- In this example the Spider would now move straight for the Bee.

```
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());
    }
}
```

Terminology

Key terms used in this lesson included:

- Abstraction
- Casting

Summary

In this lesson, you should have learned how to:

- Define abstraction and provide an example of when it is used
- Define Casting


ORACLE
ACADEMY