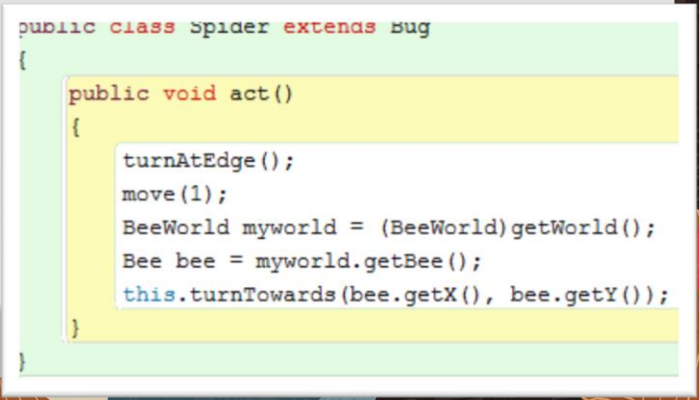
ORACLE
Academy

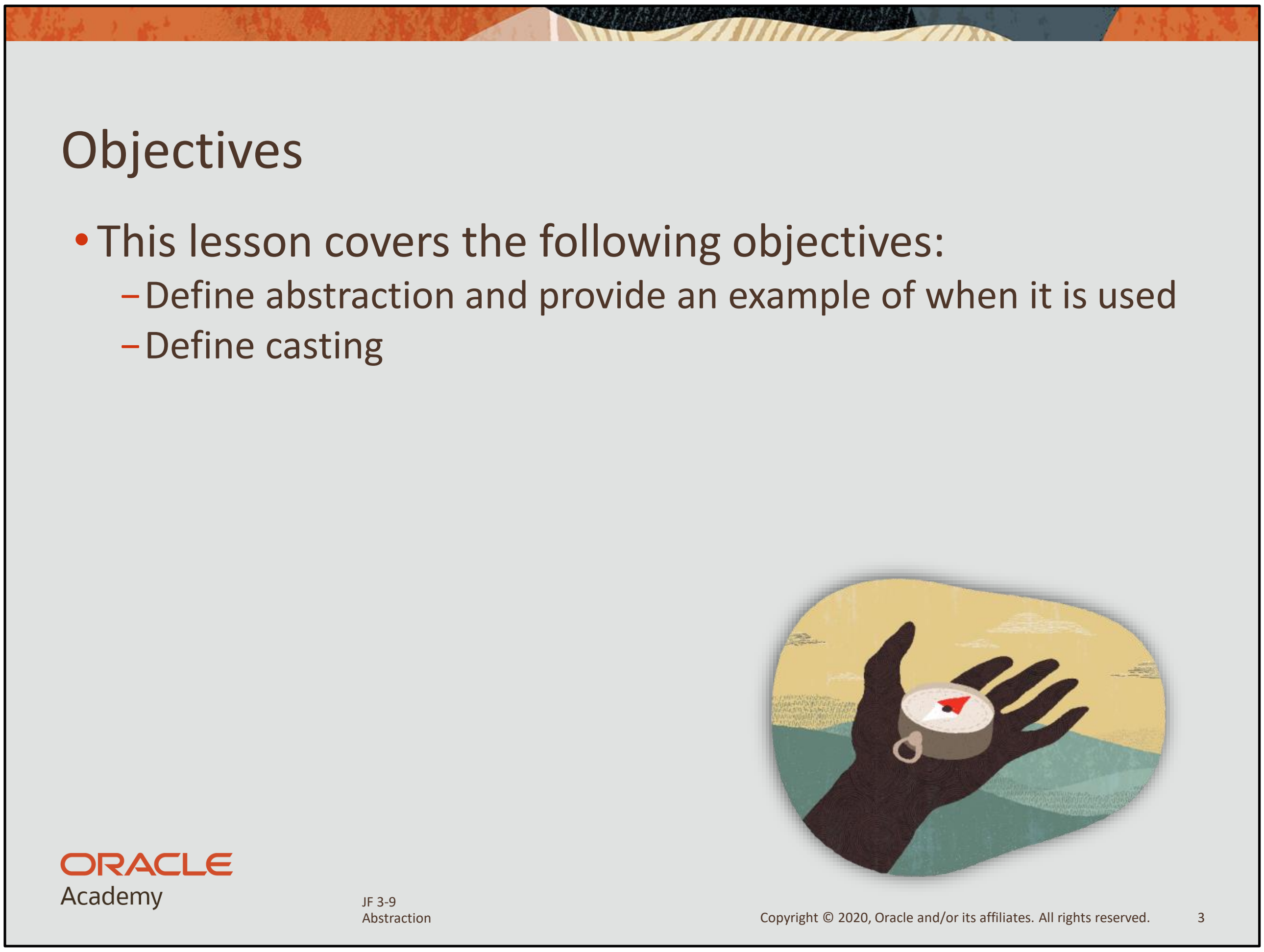
Java Fundamentals

3-9
Abstraction



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

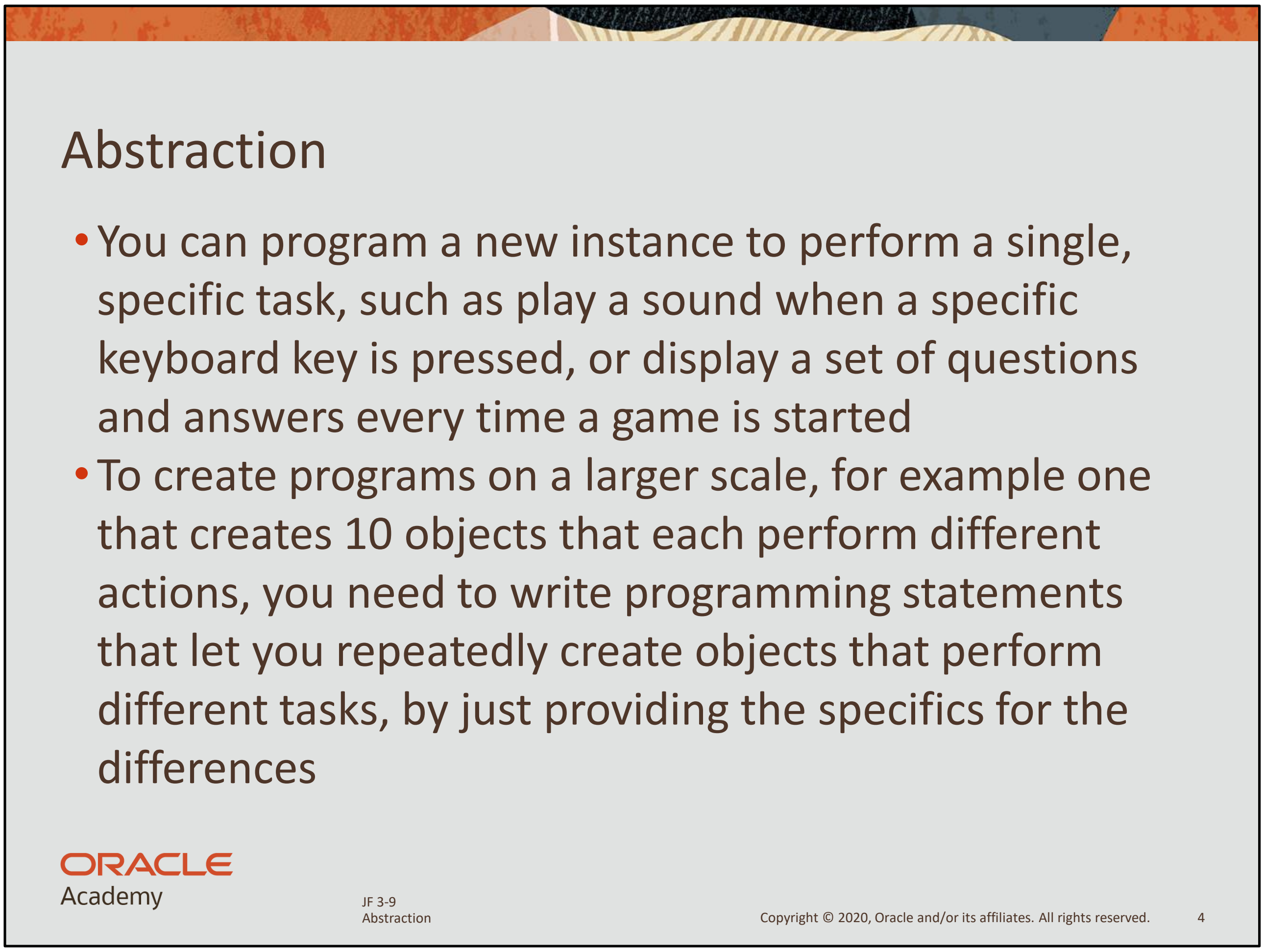
ORACLE
Academy



Objectives

- This lesson covers the following objectives:
  - Define abstraction and provide an example of when it is used
  - Define casting

ORACLE
Academy



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

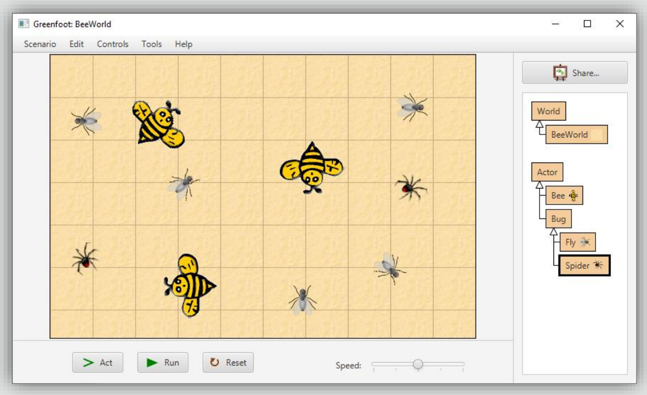
ORACLE
Academy

Abstraction can be defined in many ways. We will focus on the idea moving away from a specific function to a more general one.

Think of abstraction as the process of moving from a specific task to a more general one. So rather than calling a constructor that always does the same tasks, we could pass in values that would allow us to change the initial setup.

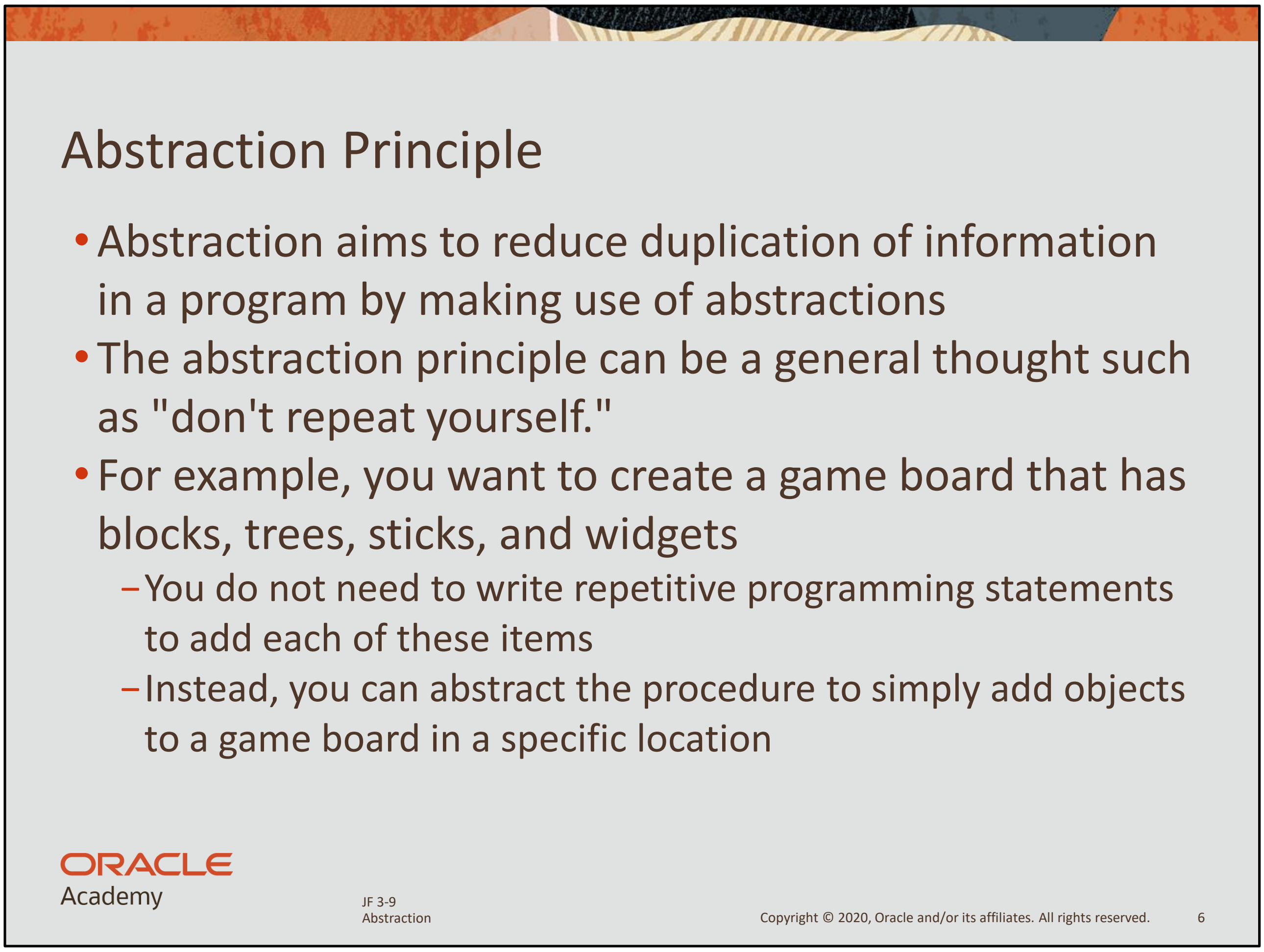
Abstraction Example

- 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



ORACLE
Academy

We have already started going down the road of abstraction by creating our own methods and constructors.



Abstraction Principle

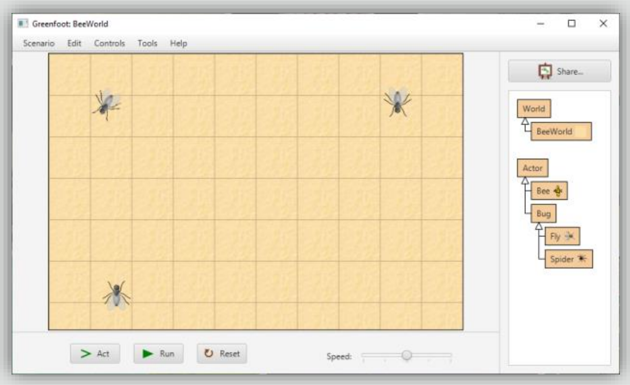
- Abstraction aims to reduce duplication of information in a program by making use of abstractions
- The abstraction principle can be a general thought such as "don't repeat yourself."
- For example, you want to create a game board that has blocks, trees, sticks, and widgets
  - You do not need to write 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

ORACLE Academy

When you write source code and find that something you are writing is very similar to other code you have written, then you should look to see whether you can abstract its purpose to another method. Then call this from the correct locations. It will then produce code that is easier to maintain.

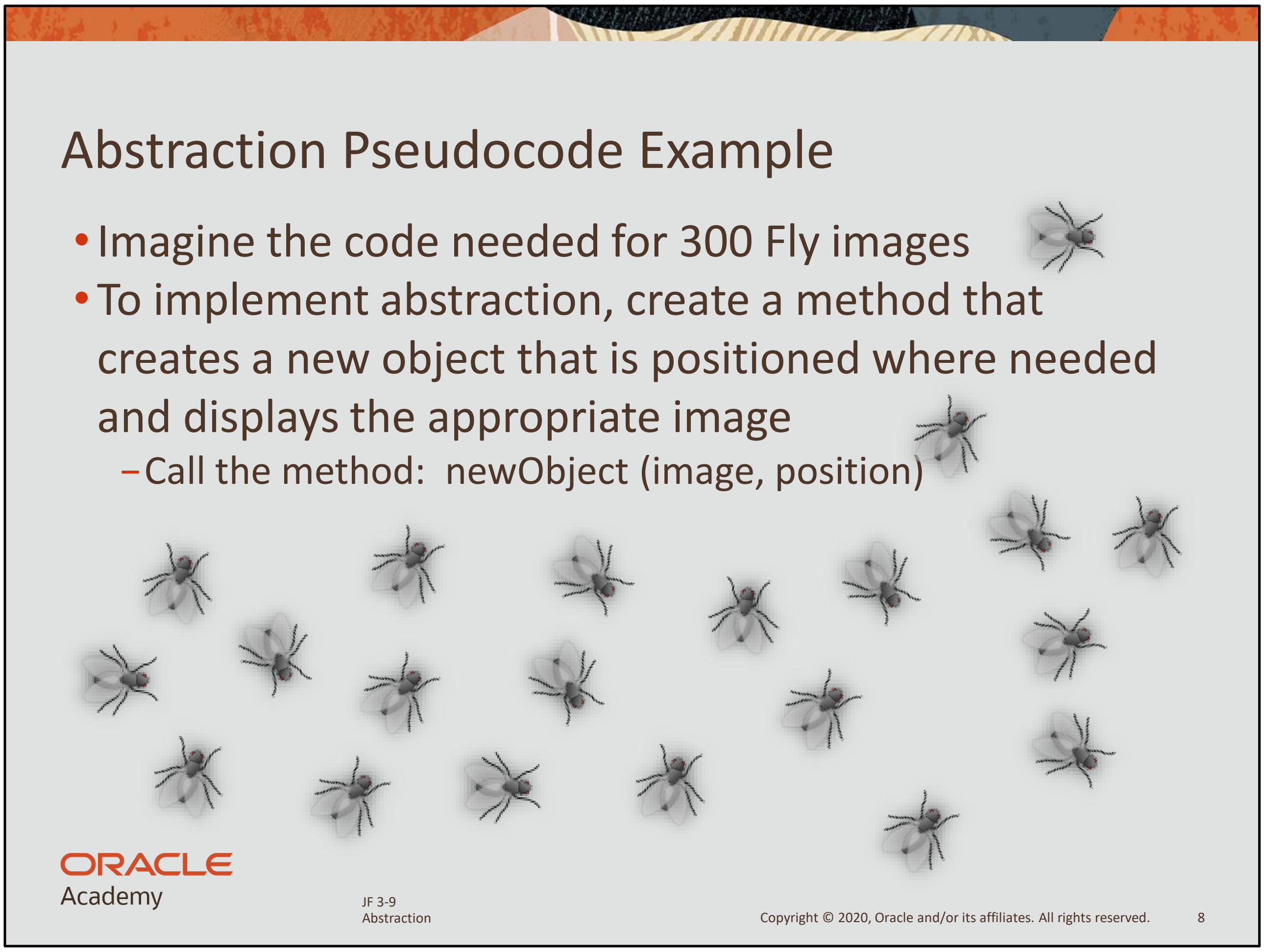
Abstraction Pseudocode Example

- 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)



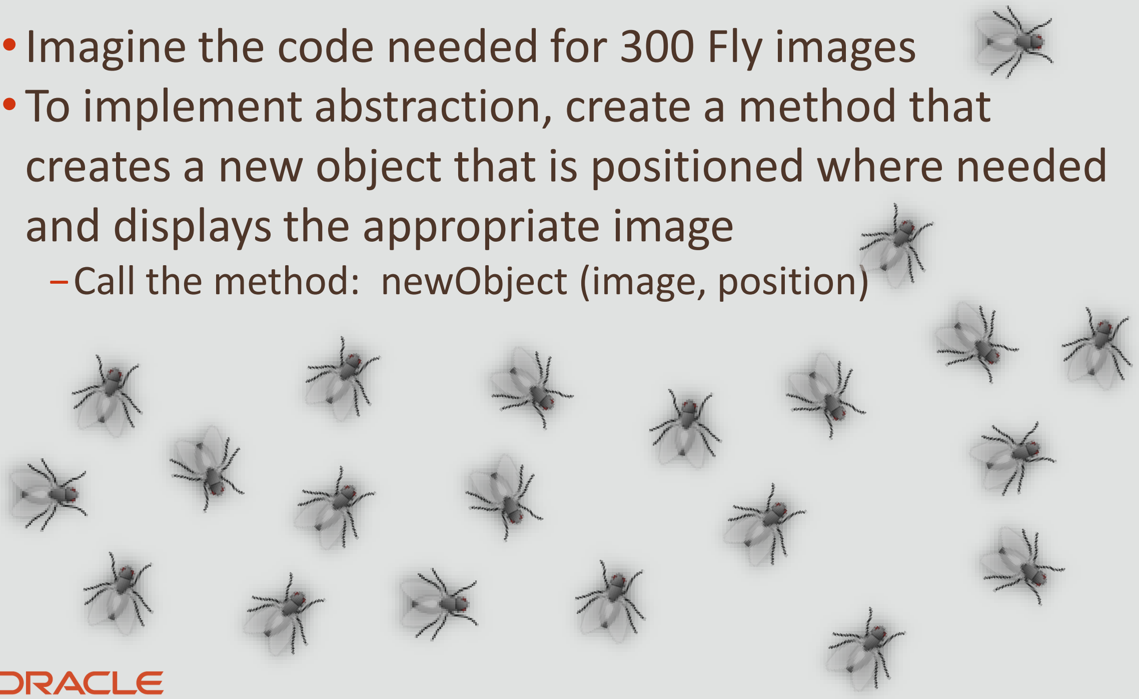
We could have written
Fly fly1 = new Fly();
fly1.setSpeed(4); // a defined method
fly1.setRotation(90);

We would have to repeat the above code for each fly. By abstracting its purpose to the constructor, then it greatly reduces repetition.



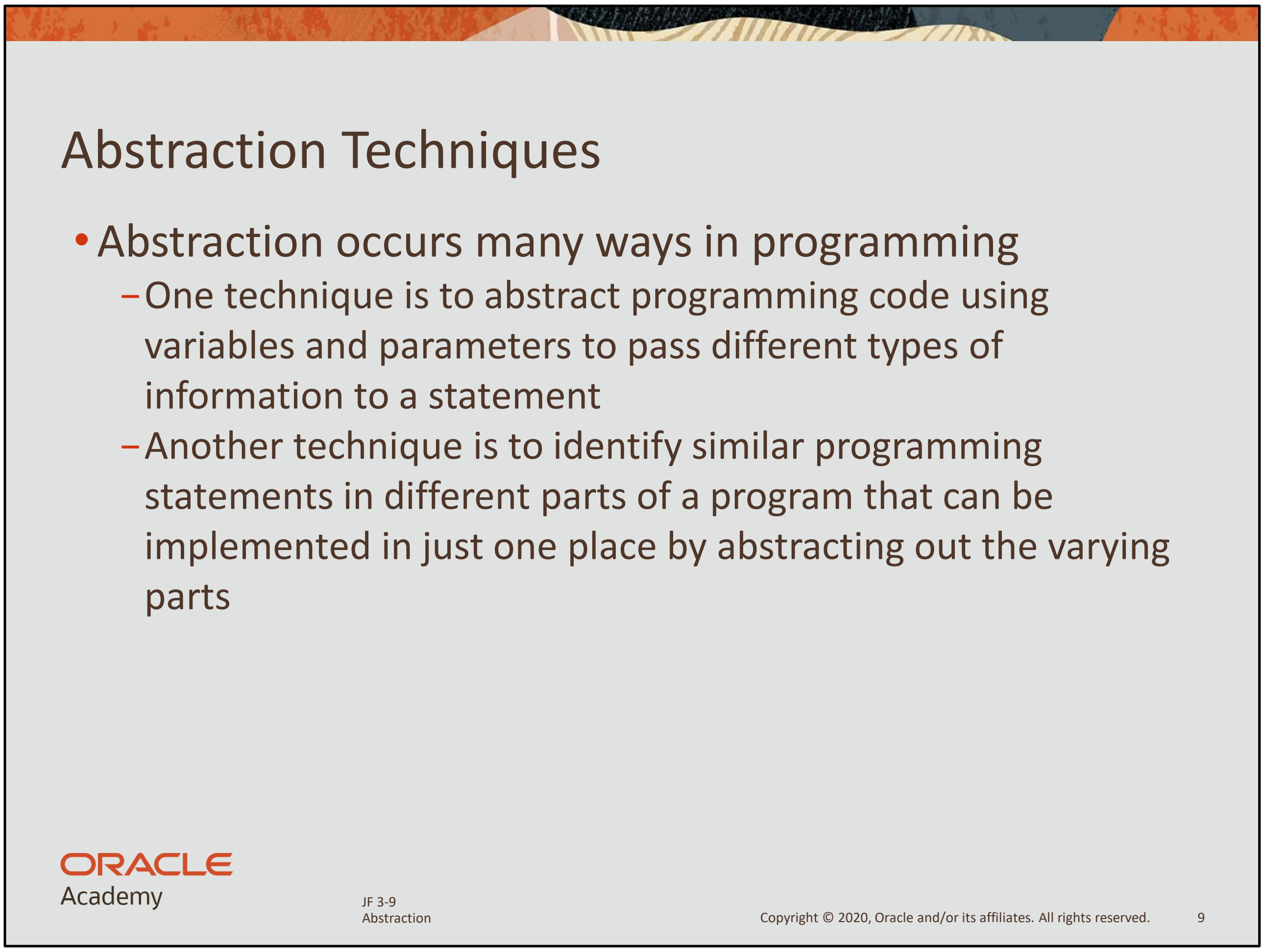
Abstraction Pseudocode Example

- 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)



ORACLE Academy

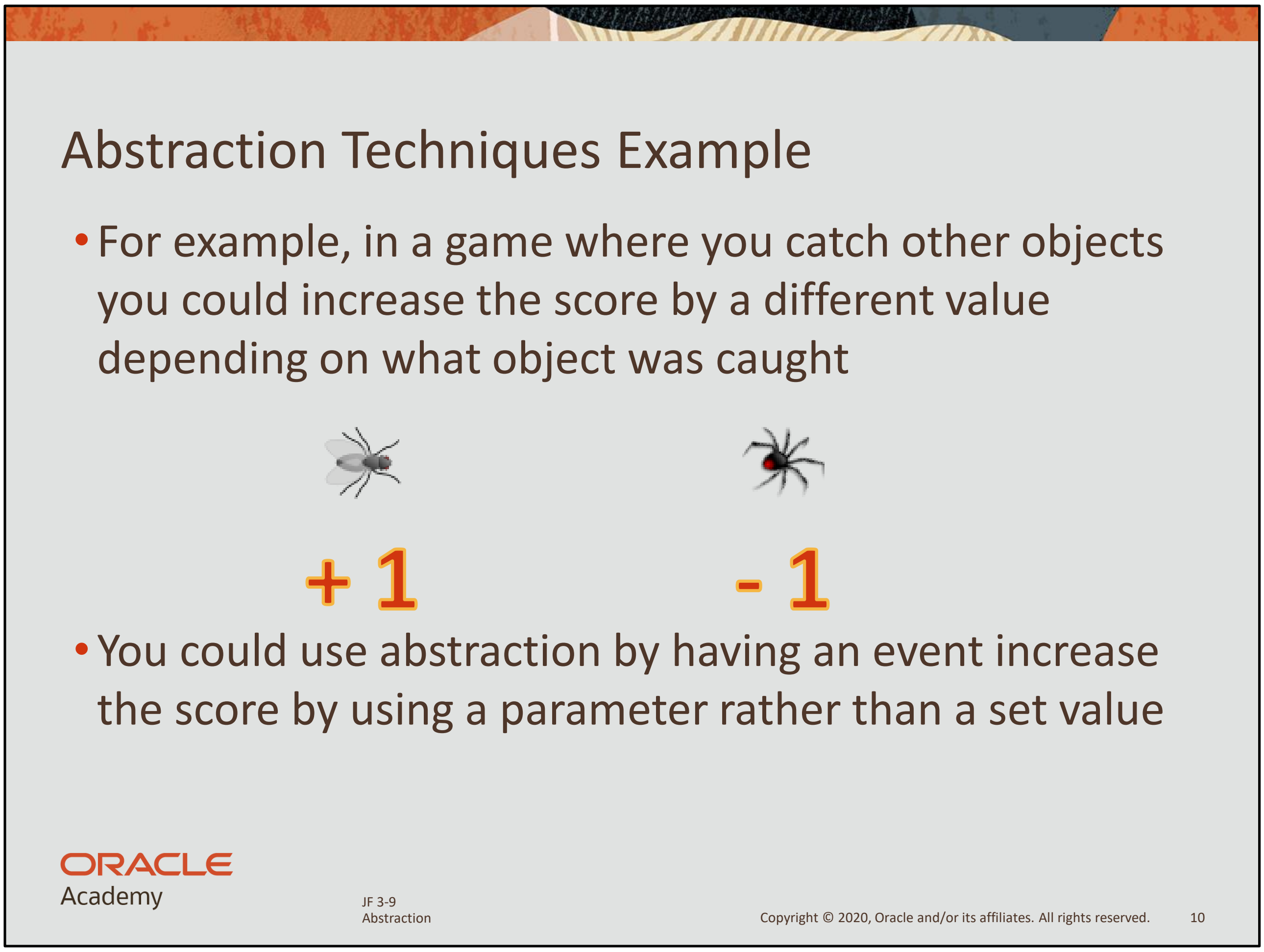
The Greenfoot programming team have created methods for you that make development easier. You can further abstract it to make it more powerful.



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

You will often modify constructors so that you can pass initial information to them.



Abstraction Techniques Example

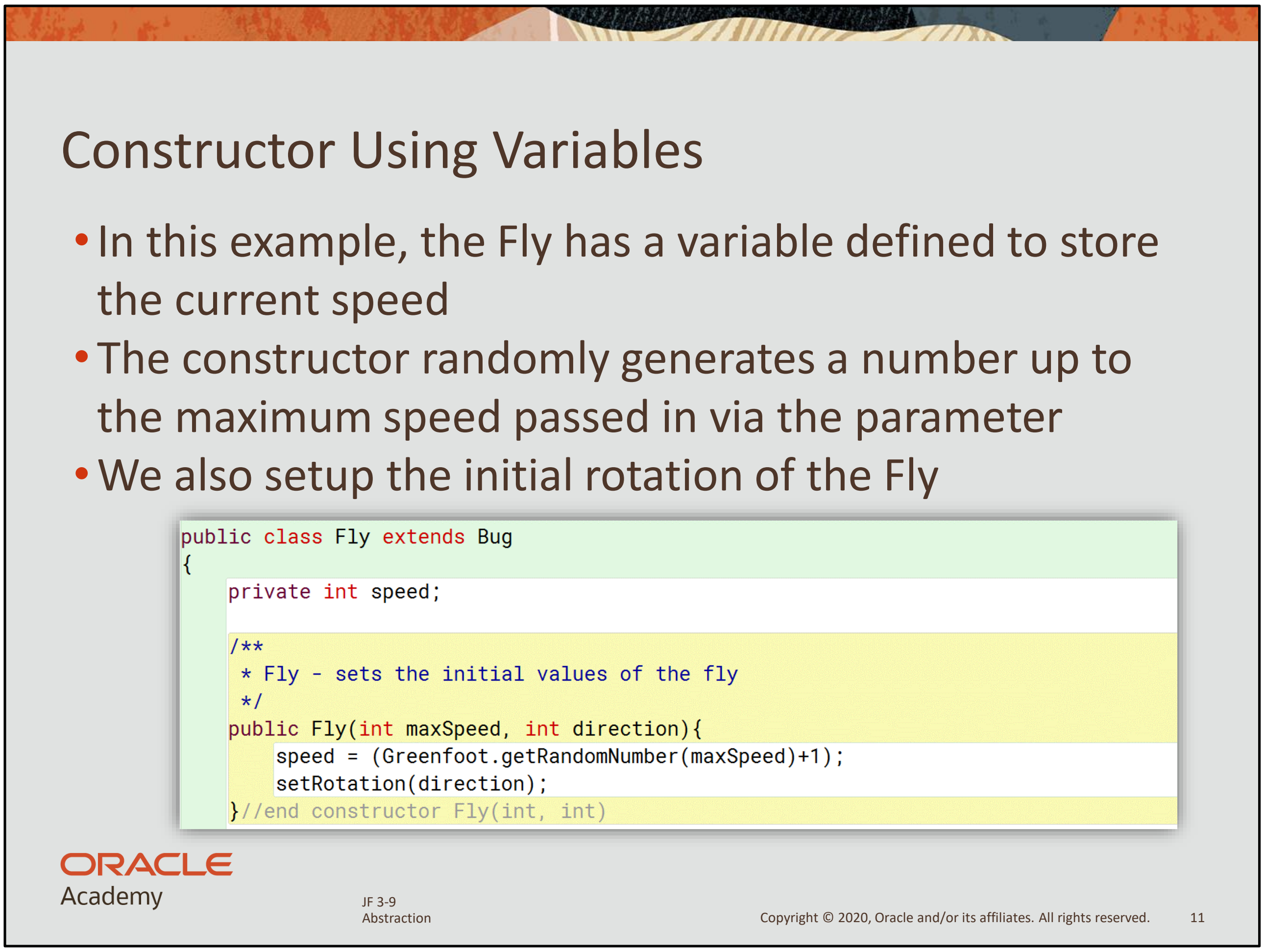
- For example, in a game where you catch other objects you could increase the score by a different value depending on what object was caught

+ 1 - 1

- You could use abstraction by having an event increase the score by using a parameter rather than a set value

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

We could call this from one area as increaseScore(5) and from another as increaseScore(10).



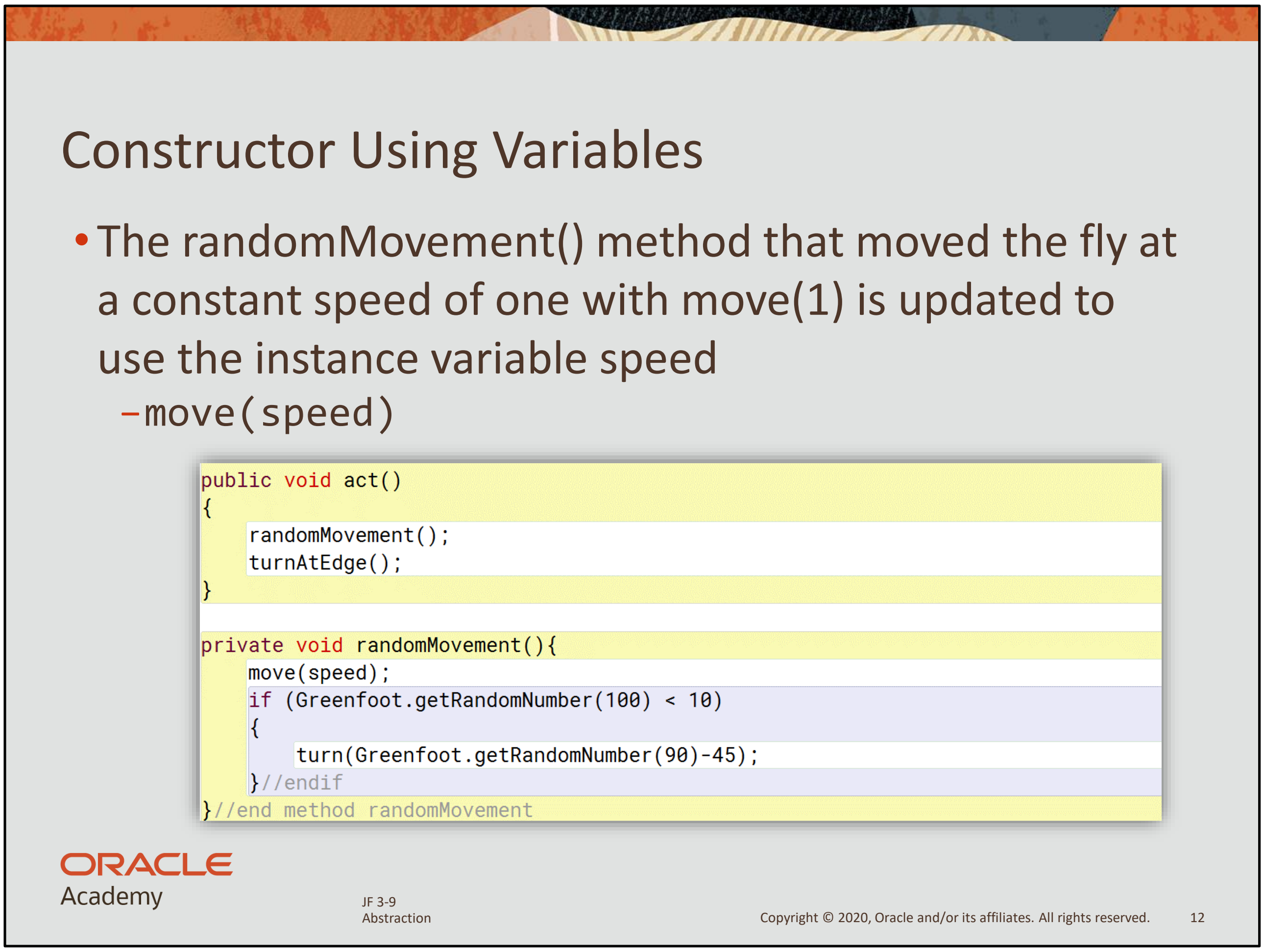
Constructor Using Variables

- 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;

    /**
     * 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)
```

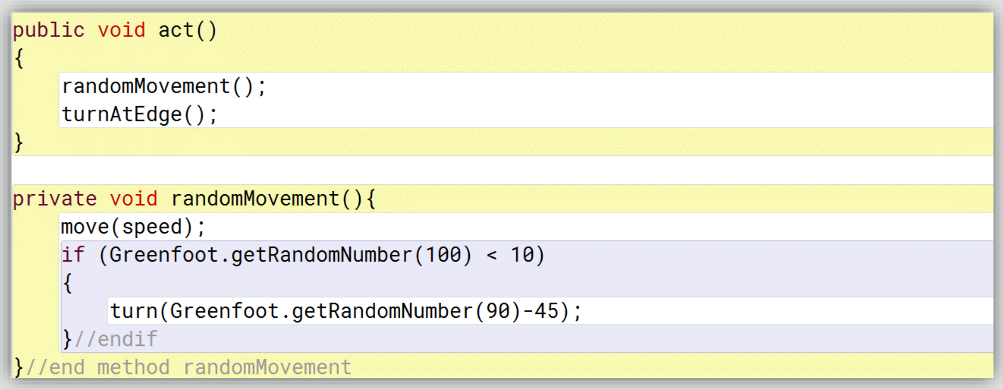
ORACLE
Academy

We add 1 to the speed because getRandomNumber() can return a 0. A fly with a speed of zero is quite easy to catch!



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)



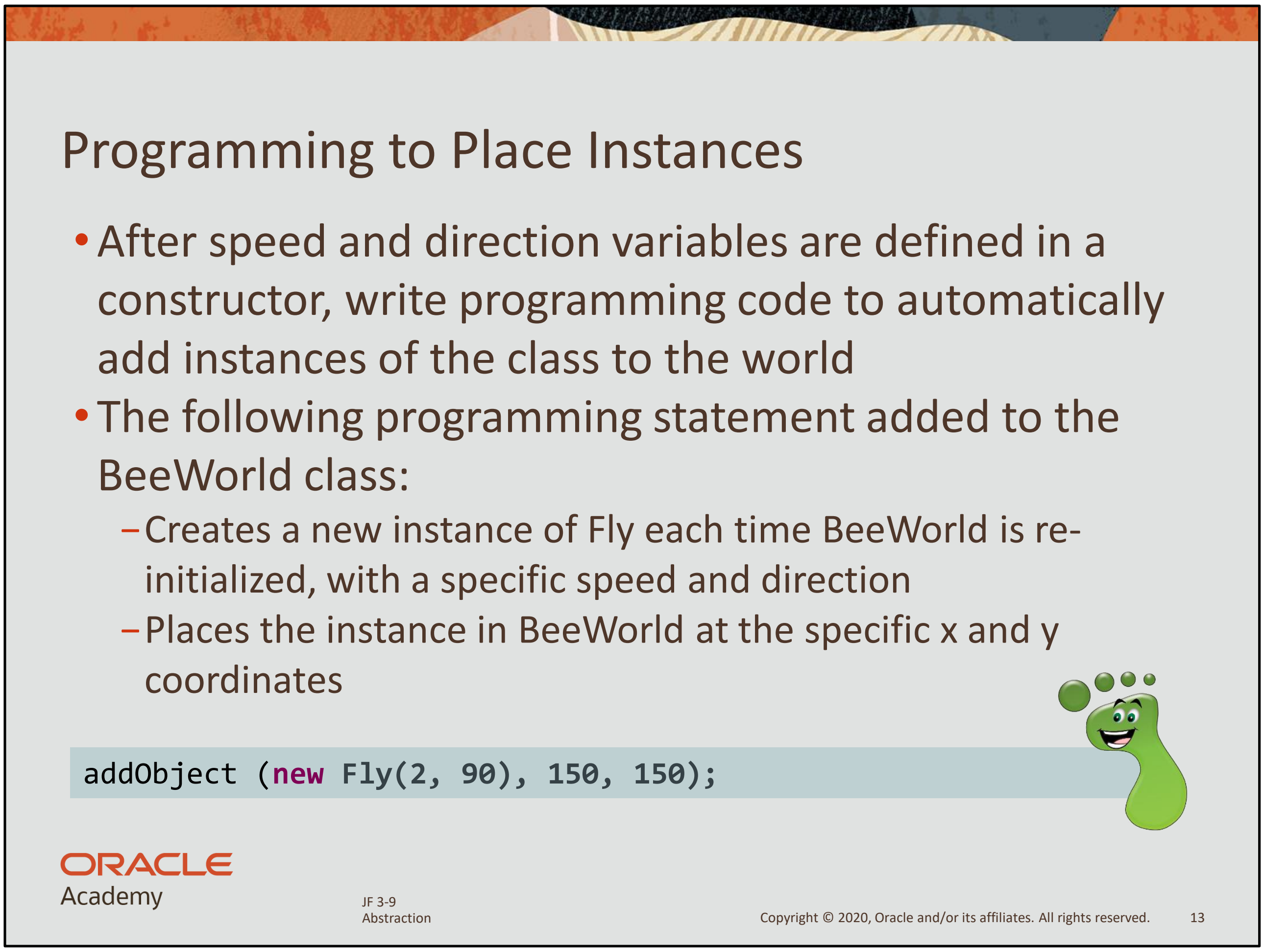
```
public void act()
{
    randomMovement();
    turnAtEdge();
}

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

ORACLE Academy

JF 3-9
Abstraction

12

If you want speed, rotation etc to change during a game then you should create a class variable to store the current value.



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);
```

ORACLE Academy

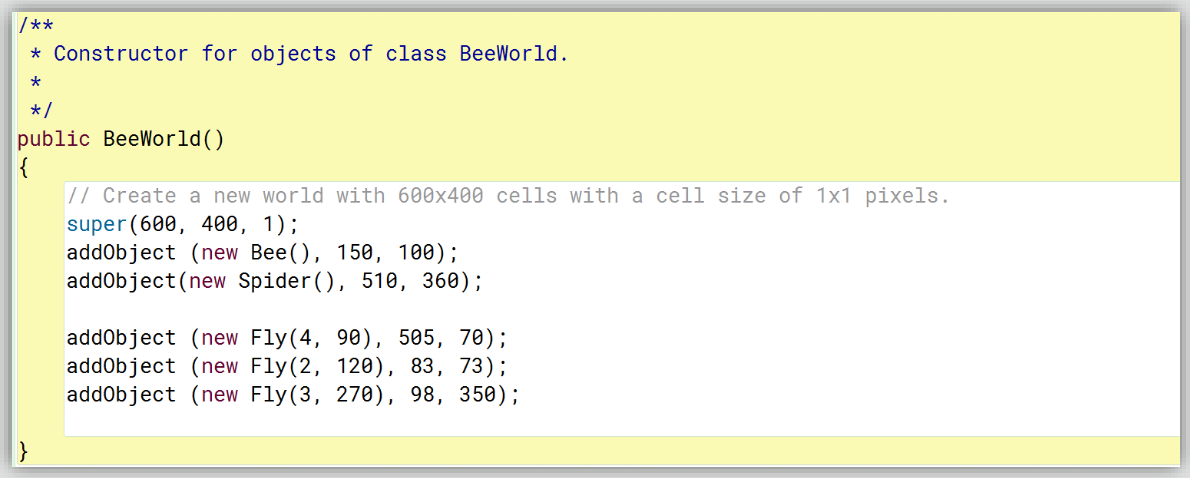
JF 3-9
Abstraction

13

This will add a Fly at coordinates (150,150) with a random maximum speed of 2 starting at a direction of 90 degrees.

Constructor Example

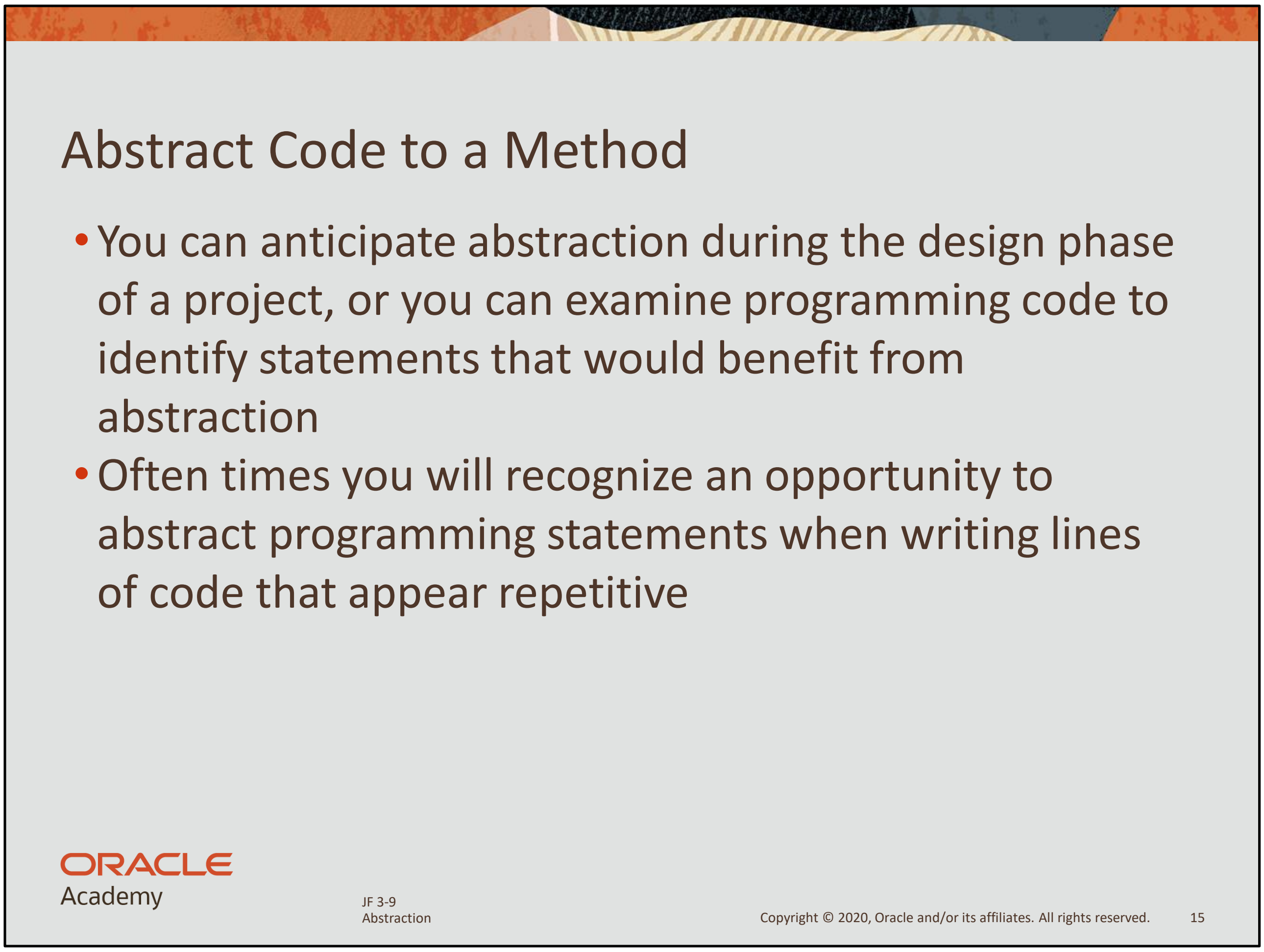
- Examine the addObject() statements in the BeeWorld constructor when adding a new Fly



```
/**
 * 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);

}
```

ORACLE
Academy

We are now adding 4 flies each with a maximum random speed, an initial direction and a different starting place. All with just 4 lines of code in the BeeWorld constructor.



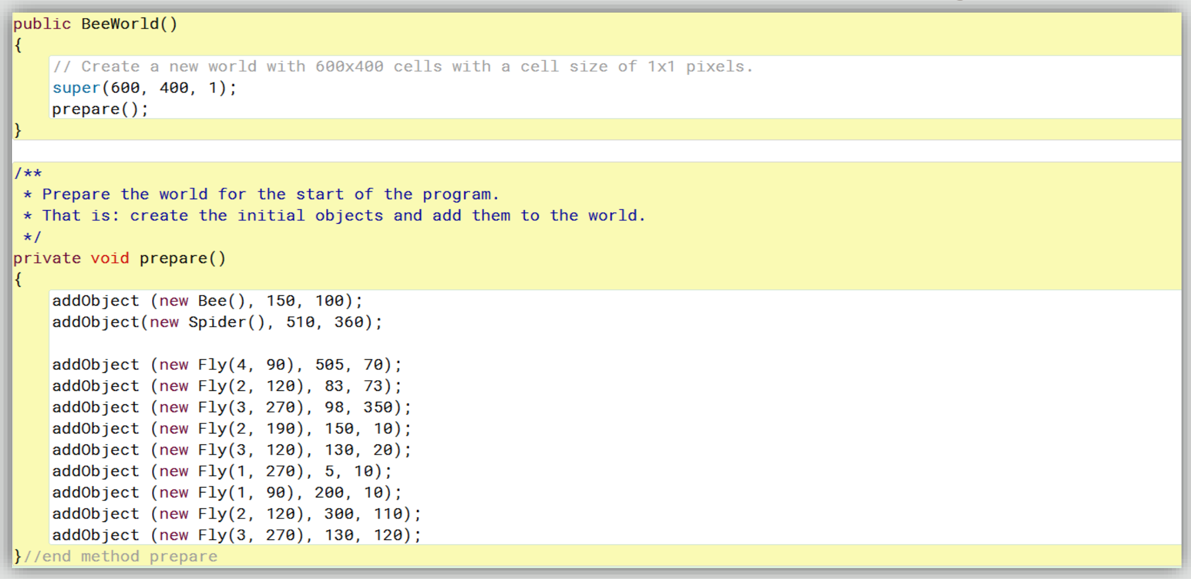
Abstract Code to a Method

- 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

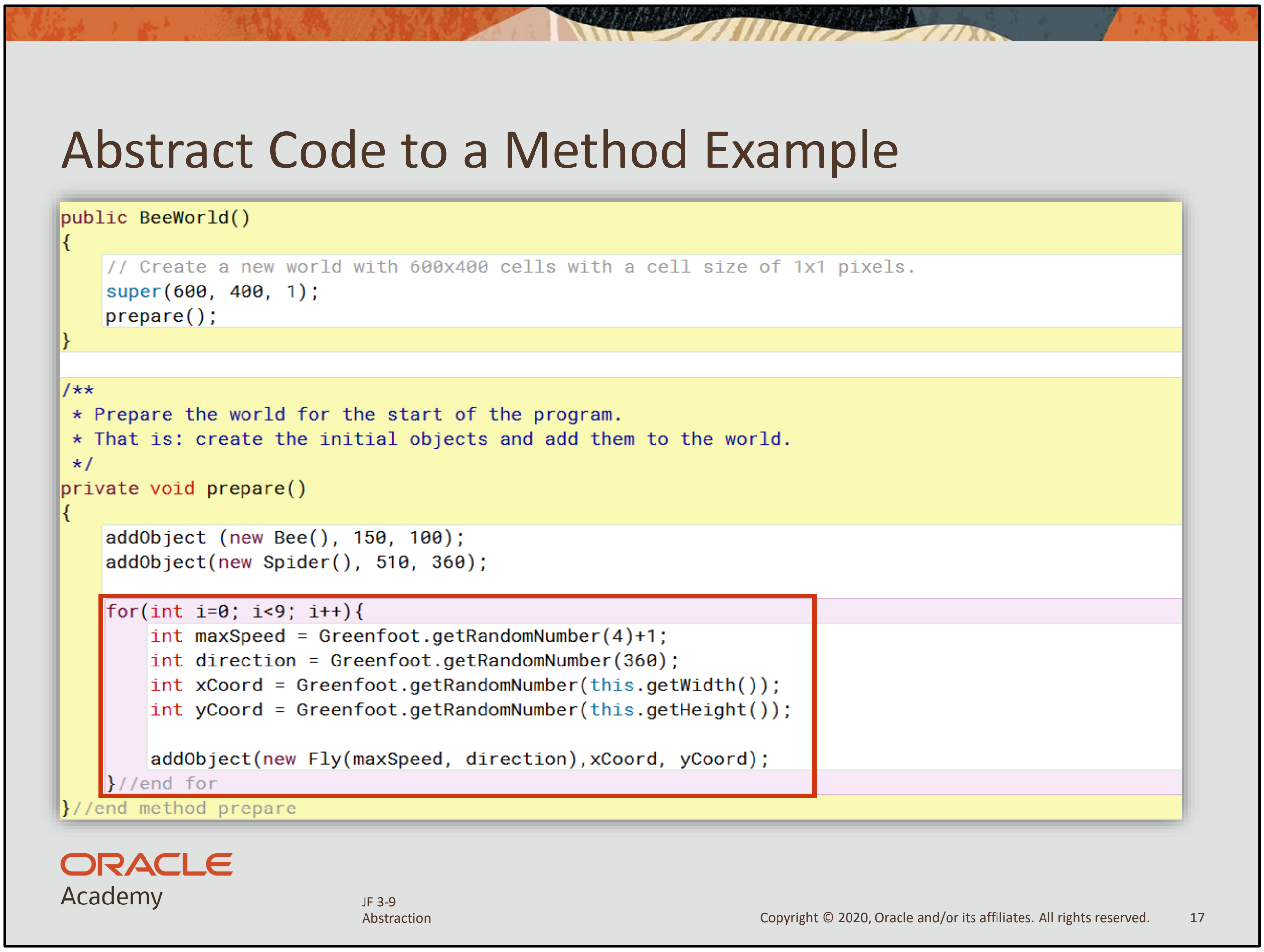
Writing lines of code that you feel you have written previously is often the trigger for abstraction to begin.

Abstract Code to a Method Example

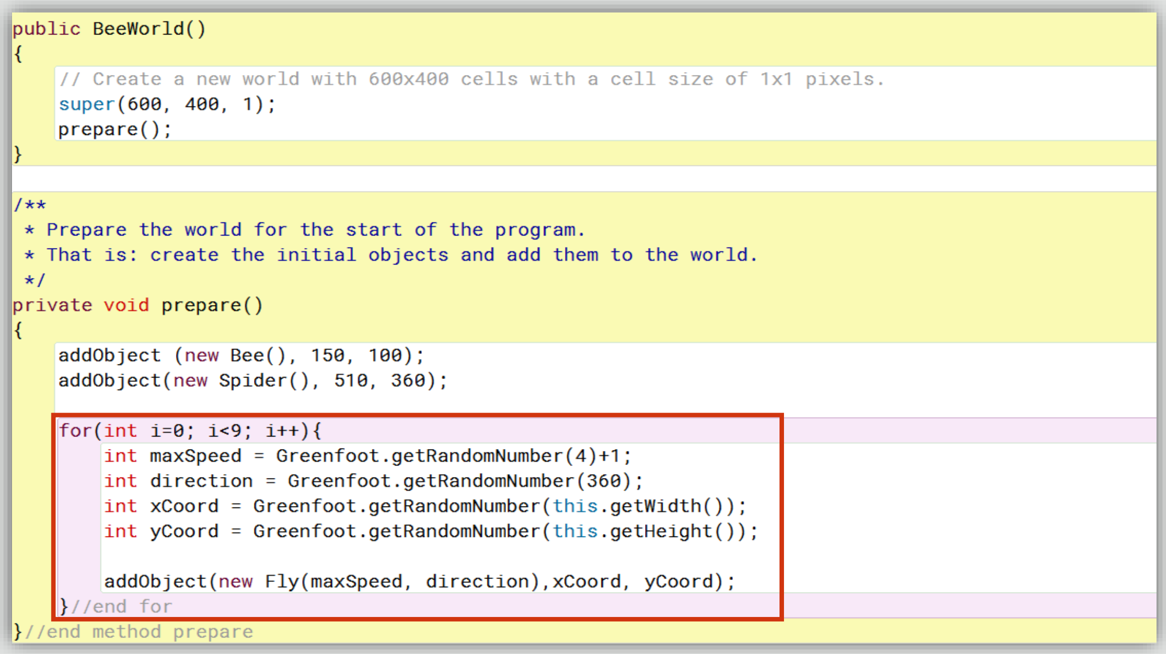
- Examine the code below and on the following slide



```
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
```



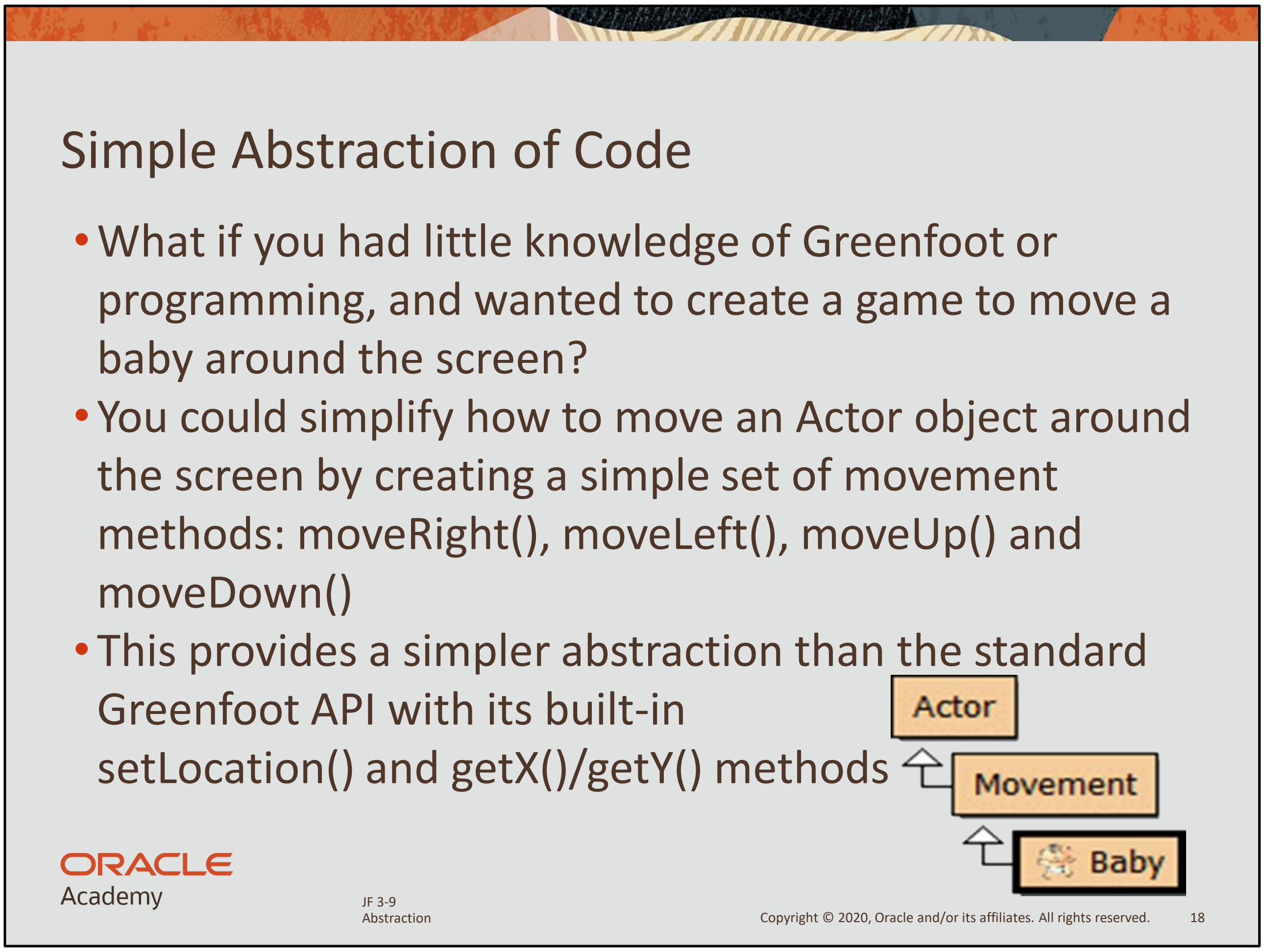
Abstract Code to a Method Example



```
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
```

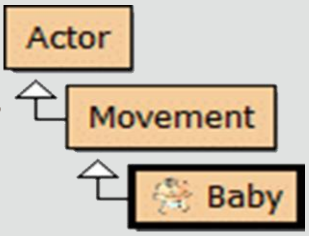
ORACLE
Academy

We create 9 flies each with a random direction and position on the screen.
This is done by using a for loop. The for loop will run the code in its brackets a set number of times. In this example it is 9 loops. We will look at loops later in the course.
We could increase the maximum speed of the flies as the game moved on to make it harder to catch them.



Simple Abstraction of Code

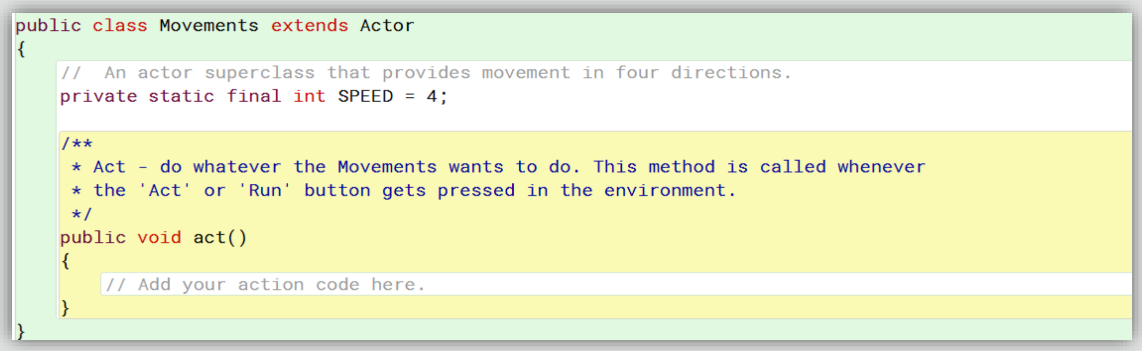
- What if you had little knowledge of Greenfoot or programming, and wanted to create a game to move a baby around the screen?
- You could simplify how to move an Actor object around the screen by creating a simple set of movement methods: moveRight(), moveLeft(), moveUp() and moveDown()
- This provides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



If you are going to have actors with shared functionality, then it is best to create a subclass of actor, add the functionality there and then subclass that class. Remember that the Movement class may have many subclasses underneath it.

Create Baby Subclass

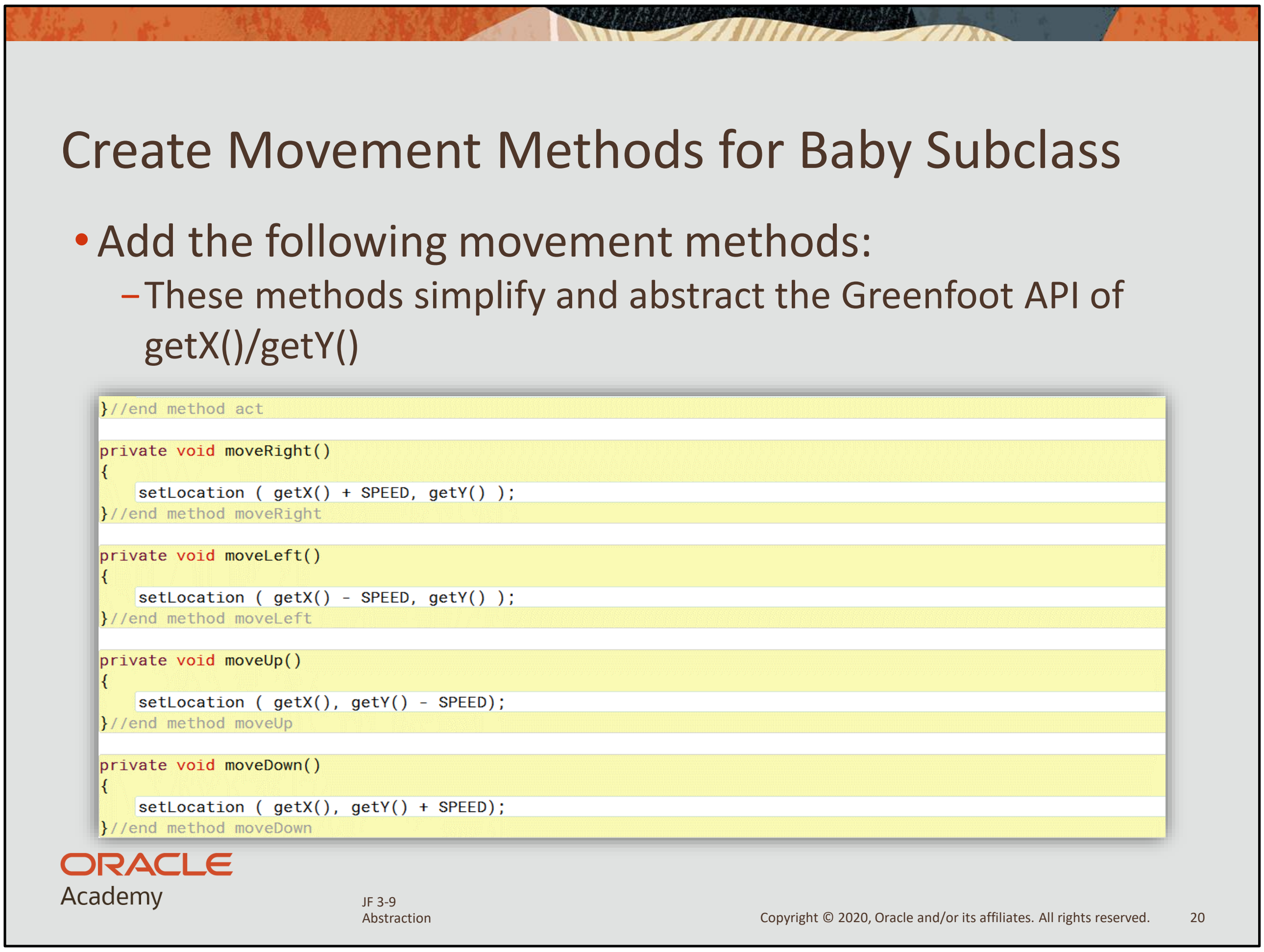
- Create a subclass of the Actor class called Movement that would allow the player to tell the Baby actor to move in a desired direction
- Add the following code to Movement which will set the amount of movement each time a move is required



```
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.
    }
}
```

ORACLE Academy

private static final int SPEED= 4
Creates a class constant that we can use in our code. Its advantage is that it makes our code more readable when we use it and also allows us to change one value to increase the speed of all of the actors that use it.



Create Movement Methods for Baby Subclass

- Add the following movement methods:
  - These methods simplify and abstract the Greenfoot API of getX()/getY()

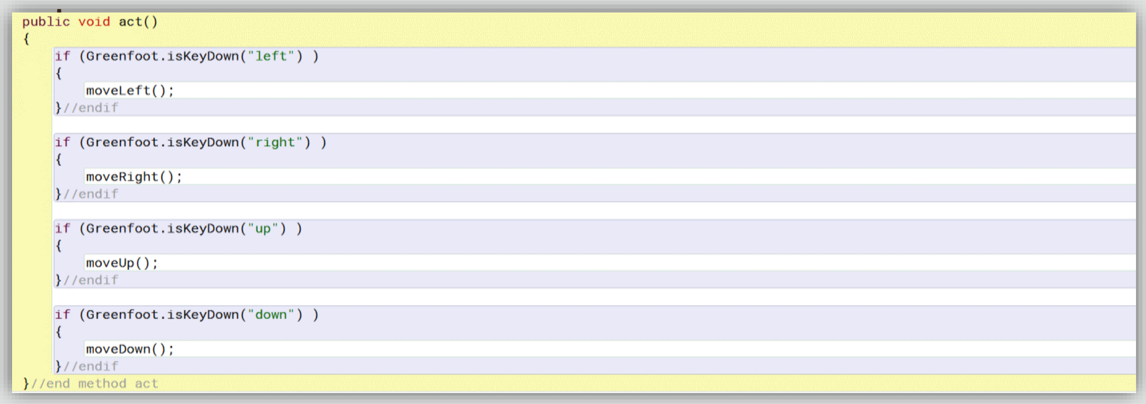
```
}//end method act

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
```

ORACLE Academy

setLocation() moves an actor without having to change its rotation. In some cases this is better than move() and rotate(). In the example we want a baby to move around. We do not want the baby to rotate, but to always stand upright, but still move around the screen in different directions.

Code the act() Method

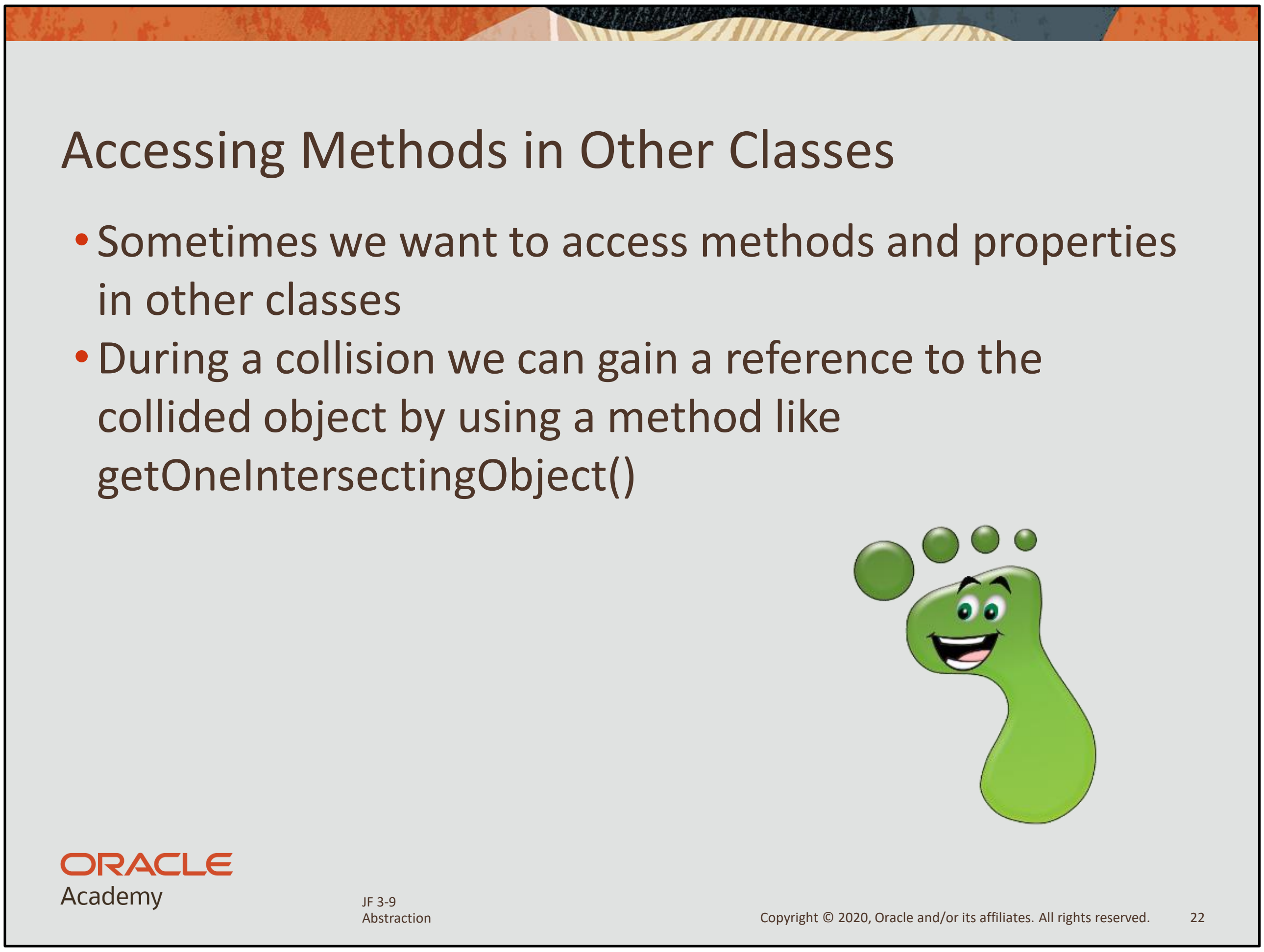
- Code the act() method of the Baby Actor class so that its instances move when the arrow keys are pressed
- This abstraction hides and automates the more complex code, only showing moveLeft(), moveRight(),



```
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
```

If the baby moved in a direction after it made contact with another actor we could call this method at this location as well.

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



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()

In some scenarios you write you will never have to worry about actions that happen away from the main actor(s), but in other games you will constantly have to monitor them.

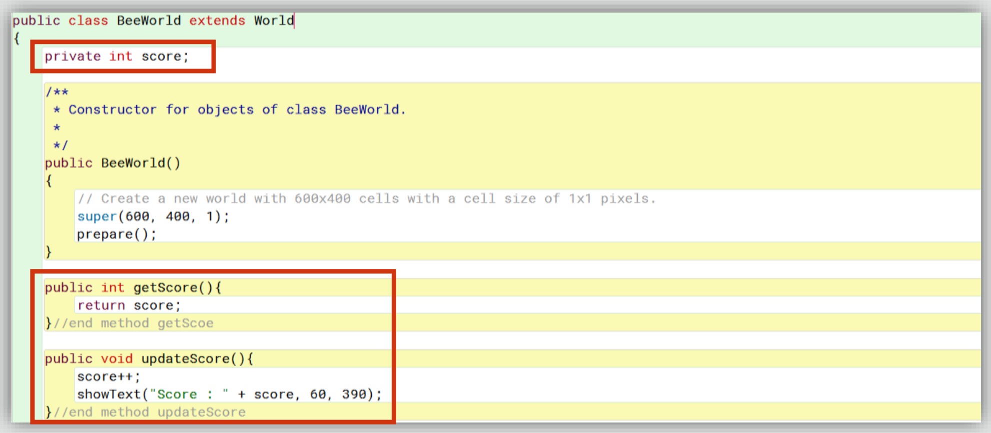
Accessing Methods in Other Classes

- 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?

In some scenarios you write you will never have to worry about actions that happen away from the main actor(s), but in other games you will constantly have to monitor them.

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



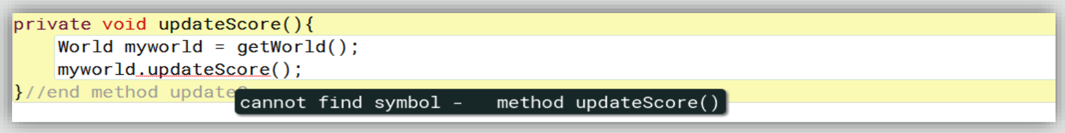
```
public class BeeWorld extends World
{
    private int score;

    /**
     * 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
```

ORACLE Academy

Notice we have kept the property score private so that no actor outside of BeeWorld has direct access. They must use the public methods getScore() and updateScore().

Accessing Other Objects Methods

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



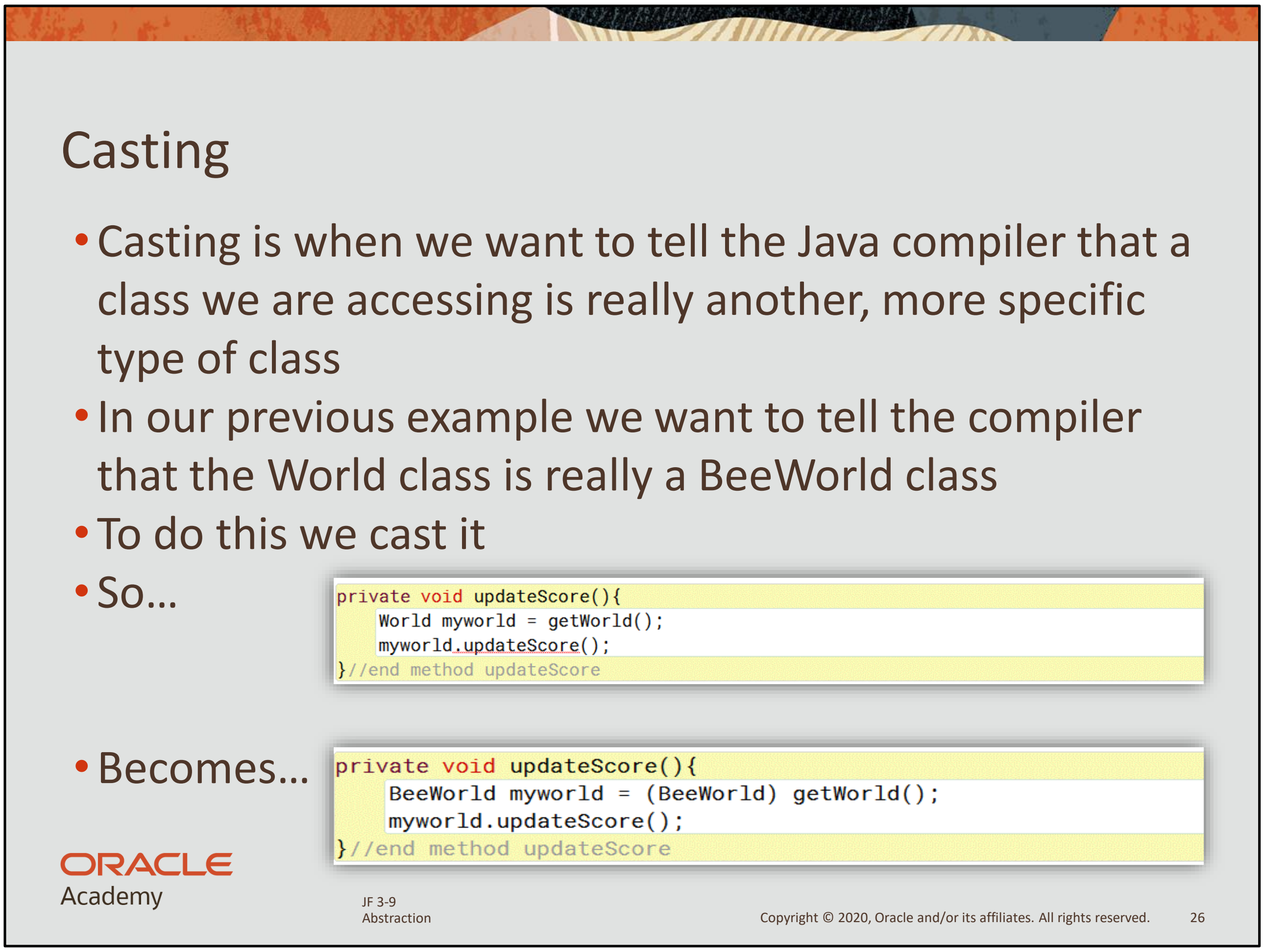
```
private void updateScore(){
    World myworld = getWorld();
    myworld.updateScore();
}//end method update
```

```
cannot find symbol -   method 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

ORACLE
Academy

Casting is a term that you will become very familiar with during this course.



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…

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

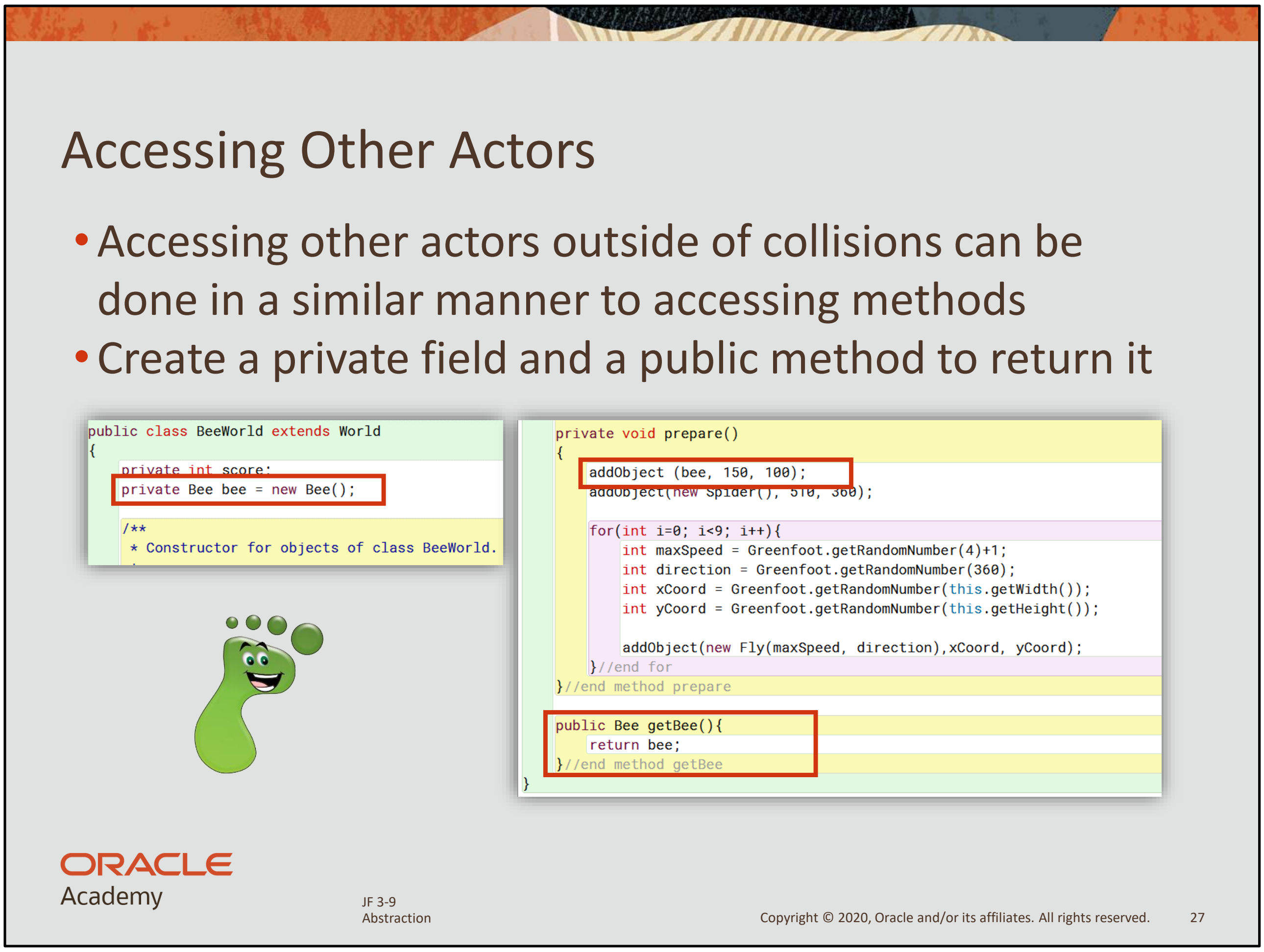
- Becomes…

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

ORACLE
Academy

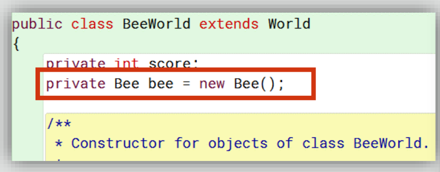
JF 3-9
Abstraction

We cannot just cast anything to become something else. There should be a relationship like World to BeeWorld or Actor to Bee. We could not cast a World to an Actor for example.

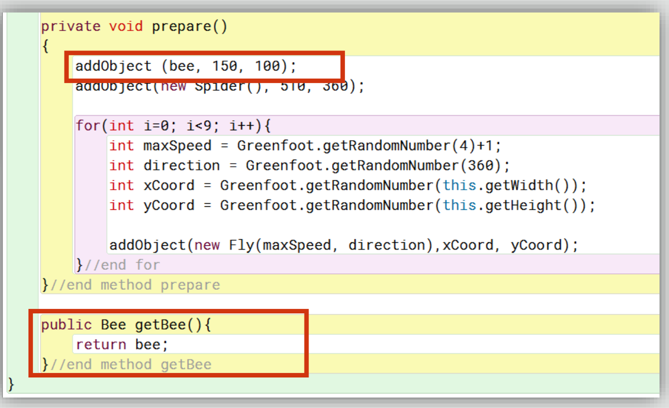


Accessing Other Actors

- 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 int score;
    private Bee bee = new Bee();

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



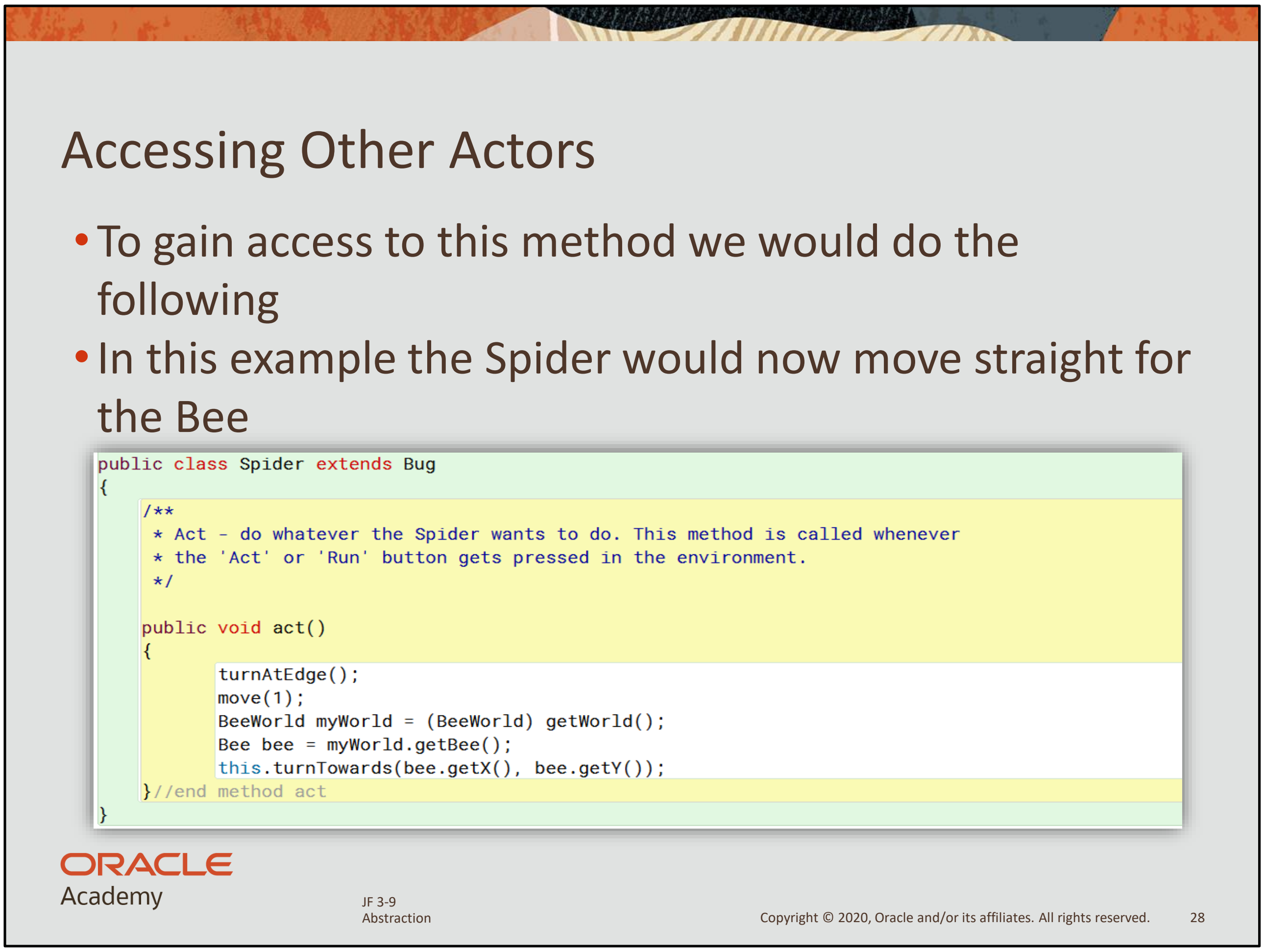
```
    private void prepare()
    {
        addObject (bee, 150, 100);
        addObject(new Spider(), 510, 360);

        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

JF 3-9
Abstraction

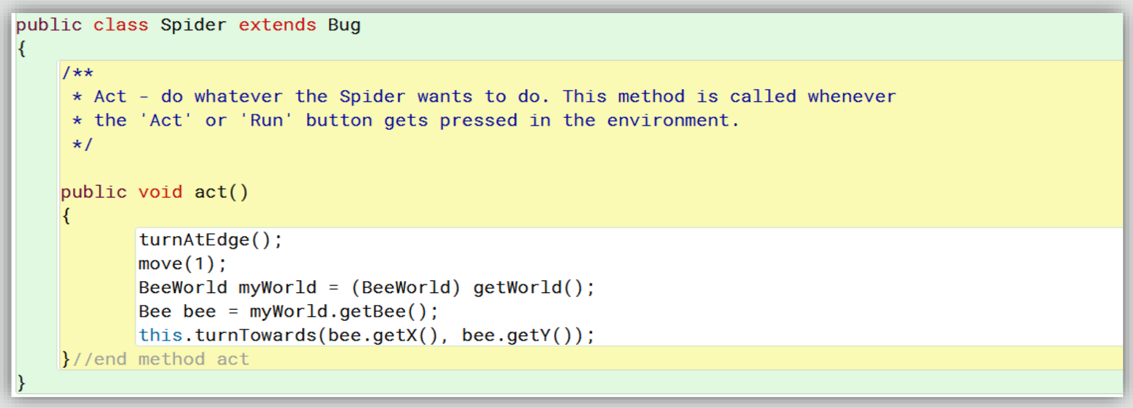
27

You should notice that this is very similar to our last example. We create a private class field to store the type of data we wish to access. Then we create a public method to return this value.



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
{
    /**
     * 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
}
```

ORACLE
Academy

BeeWorld myworld = (BeeWorld)getWorld();
This line gets a reference to the current world and stores it in the variable myworld. As getWorld's return type is World we have to cast this to a BeeWorld type.

Bee bee = myworld.getBee();
This lines create a bee variable that will reference the Bee returned by our method getBee();
Once we have this reference we can then use the returned bee to access methods of the Bee instance.
We use the getX() and getY() methods of the Bee instance in the actor method turnTowards. This will then mean the Spider will always turn to face the Bee instance in the world.

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

ORACLE
Academy