

## BUILDING AND MANIPULATING JAVA OBJECTS

An executing Java program contains a collection of objects that interact by calling each other's methods.

### *Example*

A Java program that simulates the dice game Craps might use a gambler object and two objects of class **Die**. It can model the gambler tossing the dice by having the gambler object call the **roll** method of each **Die** object.

As a Java program executes, the JVM must build the objects that are to interact, using the plans described by the classes. Some objects – such as the standard input and output objects – are automatically built by the JVM. Others must be explicitly built by statements that you, the programmer, type into the program.

### *Example*

The Java program shown on page 2 consists of two classes:

**MyApp**, which describes the application

**Die**, which describes an object that models a 6-sided game die

When the program is executed, the following happens.

Line	Runtime Operation
—	JVM builds the application object
—	JVM calls the application's <b>main</b> method
5	JVM builds a <b>Die</b> object from the plan given by class <b>Die</b>
6	JVM calls the object's <b>roll</b> method
7	JVM prints the object's <b>faceUp</b> value

```
1 public class MyApp
2 {
3     public static void main( String [] args )
4     {
5         Die bone = new Die( ); // build the die
6         bone.roll( );           // roll the die
7         System.out.println( "Roll is " + bone.faceUp );
8     }
9 }
```

```
10 // A Die object models a 6-sided die used for dice games.
11 public class Die
12 {
13     public int faceUp; // number shown facing up
14
15     public Die( )      // explicit constructor
16     {
17         roll( );
18     }
19
20     public void roll( ) // roll the die
21     {
22         // randomly select a number from 1 to 6
23         faceUp = (int)(Math.random( ) * 6 + 1);
24     }
25 }
```

## How to Build a Java Object

A Java program explicitly builds an object by executing its **new** operator and calling the class constructor. The **new** operator builds the object; the constructor initializes it. Object-oriented programmers refer to this as *instantiating* the object (since the object is an instance of the class).

A Java object is *anonymous* (i.e. it has no identifier). It is used in conjunction with a reference variable, which holds its address in memory and provides access to it. The Java program must initialize the reference variable with the address of the object as provided by **new**.

### Example

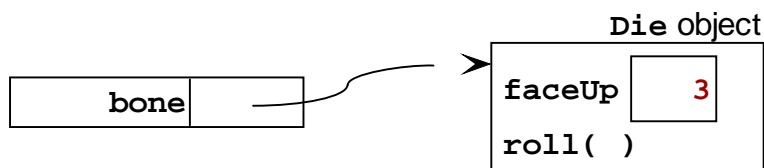
Line 5 of the **MyApp** application:

```
Die bone = new Die( );
```

Accomplishes these four operations:

1. Declares the reference variable **bone**.
2. Uses the **new** operator to build a **Die** object.
3. Calls the **Die** class's constructor to initialize the object.
4. Initializes the reference variable **bone** to the address of the object in memory.

Here's a picture of how the reference variable and object appear in memory:



## Manipulating a Java Object

Once an object is built (or instantiated) and its reference established, other objects (such as the application) can manipulate it, typically by accessing its variables and calling its methods. Any field or method whose declaration within the class is tagged **public** is accessible for manipulation. The syntax, summarized below, is different for instance variables and methods versus class variables and methods.

Syntax for Referencing an Object's Instance Variables and Methods
<i><u>reference variable identifier</u> <u>dot</u> <u>method or variable name</u></i>
<i>reference variable identifier</i> is the name of the reference variable that points to the object. <i>dot</i> refers to the period ( <b>.</b> ) symbol.

### Examples

Line 6 of the **MyApp** application above:

```
bone.roll( );
```

Calls the instance method **roll** within the object pointed to by **bone**.

Line 7 of **MyApp**:

```
System.out.println( "Roll is " + bone.faceUp );
```

Accesses and prints the **faceUp** instance variable within **bone**'s object.

Syntax for Referencing a Class's Variables and Methods
<i><u>class identifier</u> <u>dot</u> <u>method or variable name</u></i>
The syntax above represents the proper convention. If you have built an object of the class, Java also allows the following syntax, though it's not recommended:  <i><u>reference variable identifier</u> <u>dot</u> <u>method or variable name</u></i>

### Example

The class `HourlyWorker` (shown below) models employees that are paid hourly. All such workers receive the same hourly wage, which is modeled by the static field `hourlyWage`. Each worker has his or her own number of hours worked, which is modeled by the non-static field `hoursWorked`.

```
1  // An HourlyWorker object models an employee paid hourly.
2  public class HourlyWorker
3  {
4      // all workers get the same hourly wage
5      public static double hourlyWage;
6      // each worker has different hours
7      public double hoursWorked;
8
9      public static void setWage( double w )
10     // Set all workers to earn $w per hour.
11     {
12         hourlyWage = w;
13     }
14
15     public void setHours( double h )
16     {
17         hoursWorked = h;
18     }
19
20     public double grossPay( )
21     // Return worker's gross pay.
22     {
23         return hoursWorked * hourlyWage;
24     }
25 }
```

The following code creates two `HourlyWorker` objects:

```
HourlyWorker jack = new HourlyWorker( );  
HourlyWorker jill = new HourlyWorker( );
```

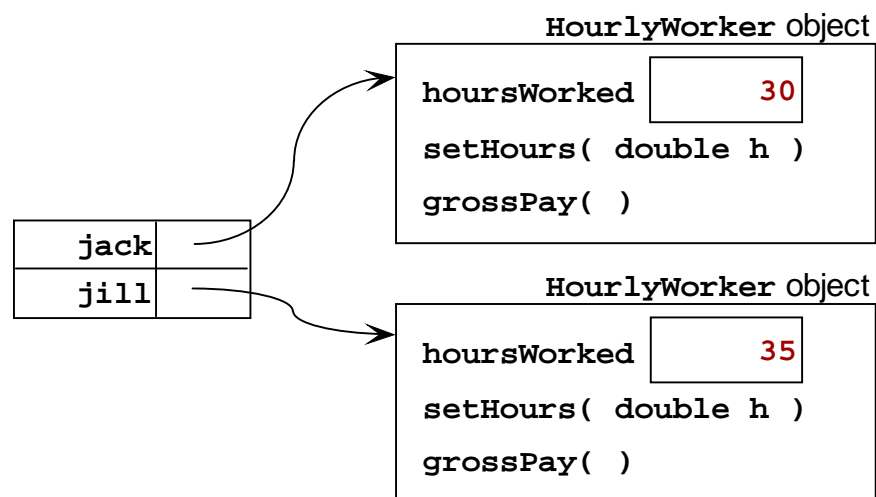
These statements set each worker's instance variable `hoursWorked`:

```
jack.setHours( 30 );  
jill.setHours( 35 );
```

And this statement sets the class variable `hourlyWage`:

```
HourlyWorker.setWage( 10.50 );
```

Here's a picture of the objects in memory:



### Class Variables and Methods

```
HourlyWorker.hourlyWage 10.50  
HourlyWorker.setWage( double w )
```

The following statement will also work to set the class variable `hourlyWage`:

```
jill.setWage( 10.50 );
```

A reference variable that doesn't refer to any object is called a *null pointer*; its value is denoted by the keyword `null`. If you use a null pointer to attempt to reference an object before the object has been built, you'll get a run-time error.

### *Example*

These Java statements

```
Die bone;  
bone.roll( );
```

When executed result in this run-time error:

```
Exception in evaluation thread java.lang.NullPointerException
```

## Exercises

Enter class **Die** into jGRASP and save it to a file. Compile it. Perform the following experiments using jGRASP's Interactions pane. To answer the questions, observe jGRASP's Workbench pane.



1. In jGRASP's Interactions pane, enter and execute the statement:

```
Die bone;
```

In jGRASP's Workbench pane, what is the value of reference variable **bone**?

2. In jGRASP's Interactions pane, enter and execute the statement:

```
bone = new Die( );
```

In jGRASP's Workbench pane, what is the value of reference variable **bone**? Find the entry for **bone** and click the  so that it becomes . What is the value of the object's instance variable **faceUp**?

3. In jGRASP's Interactions pane, enter and execute the statement:

```
bone.roll( );
```

In jGRASP's Workbench pane, what is the value of the object's instance variable **faceUp**?

4. In jGRASP's Interactions pane, enter and execute the statement:

```
System.out.println( bone.faceUp );
```

What value is displayed?

5. In jGRASP's Interactions pane, enter and execute the statement:

```
Die bone2 = new Die( );
```

In jGRASP's Workbench pane, what is the value of reference variable **bone2**? What is the value of the **bone2** object's instance variable **faceUp**?



Enter class <b>HourlyWorker</b> into jGRASP and save it to a file. Compile it. Perform the following experiments using jGRASP's Interactions pane. To answer the questions, observe jGRASP's Workbench pane.	
6.	<p>In jGRASP's Interactions pane, enter and execute the statement:</p> <pre><b>HourlyWorker jane;</b></pre> <p>In jGRASP's Workbench pane, what is the value of reference variable <b>jane</b>?</p>
7.	<p>In jGRASP's Interactions pane, enter and execute the statement:</p> <pre><b>jane = new HourlyWorker( );</b></pre> <p>In jGRASP's Workbench pane, what is the value of reference variable <b>jane</b>? Find the entry for <b>jane</b> and click the  so that it becomes . What is the value of <b>jane</b>'s instance variable <b>hoursWorked</b>?</p>
8.	<p>In jGRASP's Interactions pane, enter and execute the statement:</p> <pre><b>jane.setHours( 25 );</b></pre> <p>In jGRASP's Workbench pane, what is the value of <b>jane</b>'s instance variable <b>hoursWorked</b>?</p>
9.	<p>In jGRASP's Interactions pane, enter and execute the statements:</p> <pre><b>HourlyWorker dick = new HourlyWorker( );</b> <b>dick.setHours( 20 );</b></pre> <p>In jGRASP's Workbench pane, what is the value of reference variable <b>dick</b>? Find the entry for <b>dick</b> and click the  so that it becomes . What is the value of <b>dick</b>'s instance variable <b>hoursWorked</b>?</p>
10.	<p>In jGRASP's Interactions pane, enter and execute the statement:</p> <pre><b>HourlyWorker.setWage( 12 );</b></pre> <p>In jGRASP's Workbench pane, what is the value of the class variable <b>hourlyWage</b>?</p>

11.	<p>In jGRASP's Interactions pane, enter and execute the statement:</p> <pre>jane.setWage( 10 );</pre> <p>Look at jGRASP's Workbench pane. What happened? Why?</p>
12.	<p>In jGRASP's Interactions pane, enter and execute the statement:</p> <pre>HourlyWorker.setHours( 40 );</pre> <p>What happened? Why?</p>
13.	Enter into the <i>Interactions</i> pane the correct Java statement that will display Jane's pay.
14.	Enter into the <i>Interactions</i> pane the correct Java statement that will display Dick's pay.

Each code fragment below tries to build an object of class **Die**. For each, circle what's wrong with the code and explain. One of them is correct.

15.	<b>Die bone;</b>
16.	<b>Die bone = new Die;</b>
17.	<b>Die bone = Die( );</b>
18.	<b>bone = new Die( );</b>
19.	<b>die bone = new die( );</b>
20.	<b>Die bone = new Die( );</b>

Explain the error in each of the following applications. None of them is correct.

21. 

```
public class MyApp
{
    public static void main( String [] args )
    {
        Die.roll( );
        System.out.println( "Roll is " + Die.faceUp );
    }
}
```

22. 

```
public class MyApp
{
    public static void main( String [] args )
    {
        bone.roll( );
        System.out.println( "Roll is " + bone.faceUp );
    }
}
```

23. 

```
public class MyApp
{
    public static void main( String [] args )
    {
        Die bone;
        bone.roll( );
        System.out.println( "Roll is " + bone.faceUp );
    }
}
```

Assume that **bone** refers to a correctly built **Die** object. Each statement below tries to call its **roll** method. Circle what's wrong with the code and explain. One of them is correct.

24. **Bone.roll( );**

25. **bone.Roll( );**

26. **roll( );**

27. **bone.roll;**

28. **bone.roll( );**

Assume that **bone** refers to a correctly built **Die** object. Each statement below tries to access its instance variable **faceUp**. Circle what's wrong with the code and explain. One of them is correct.

29. **System.out.println( "Roll is " + Bone.faceUp );**

30. **System.out.println( "Roll is " + bone.FaceUp );**

31. **System.out.println( "Roll is " + faceUp );**

32. **System.out.println( "Roll is " + bone.faceUp( ) );**

33. **System.out.println( "Roll is " + bone.faceUp );**

Assume that **bill** refers to a correctly built **HourlyWorker** object. Each statement below tries to call one of the **HourlyWorker** methods. Circle what's wrong with the code and explain. Two of them are correct.

34. **bill.setWage( 9.5 );**

35. **Worker.setWage( 9.5 );**

36. **bill.setHours( 40 );**

37. **Worker.setHours( 40 );**