B.Sc. In Software Development.

Applications Programming

Introduction to OO.

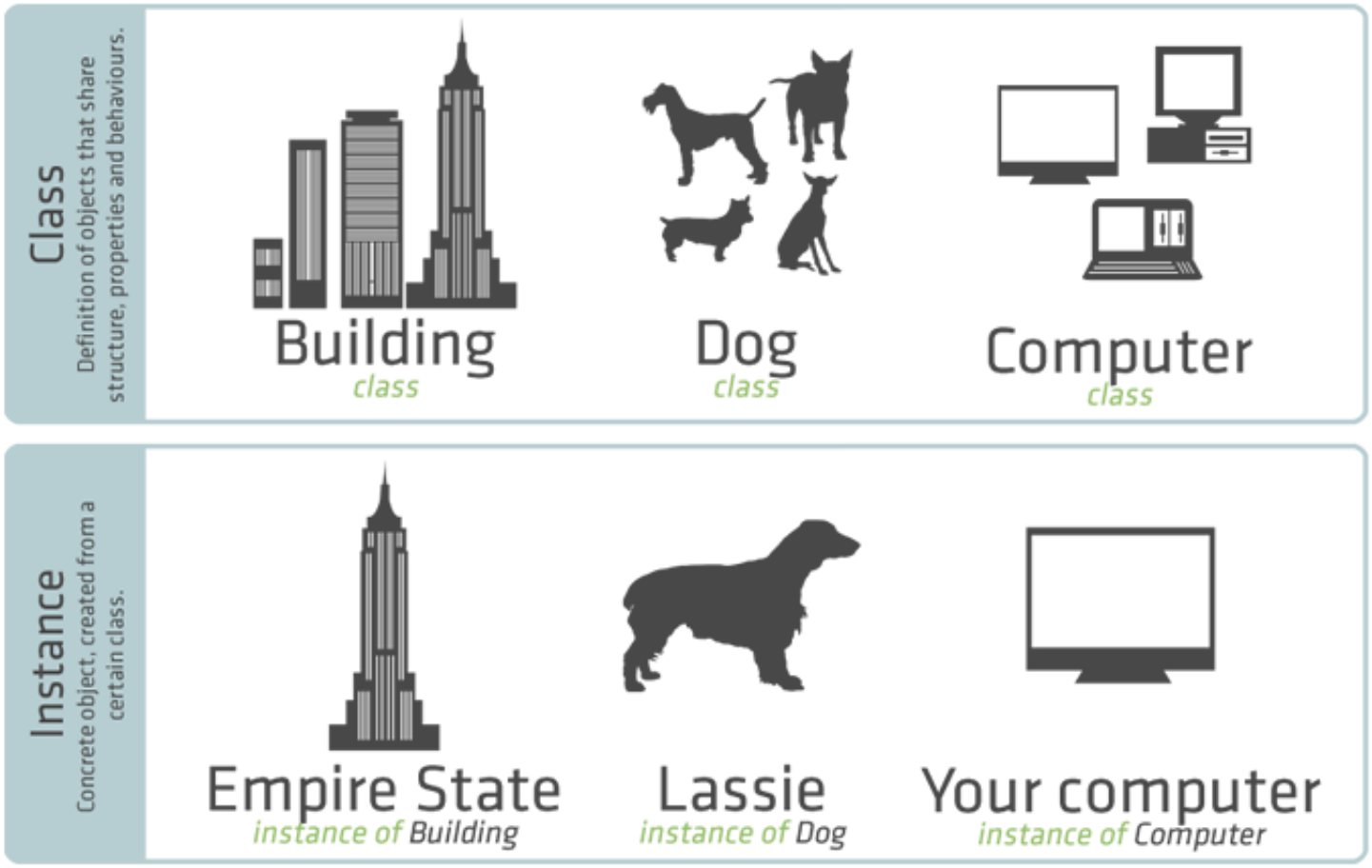
LIMERICK INSTITUTE
OF TECHNOLOGY
SCHOOL OF SCIENCE,
ENGINEERING & I.T.
Department of Information Technology

Introduction

- OO is based on simulation and modelling
  - Trace its roots back to the mid 1960's in Norway.
  - Pioneered by Kirsten Nygaard and Loe-Johan Dhal.

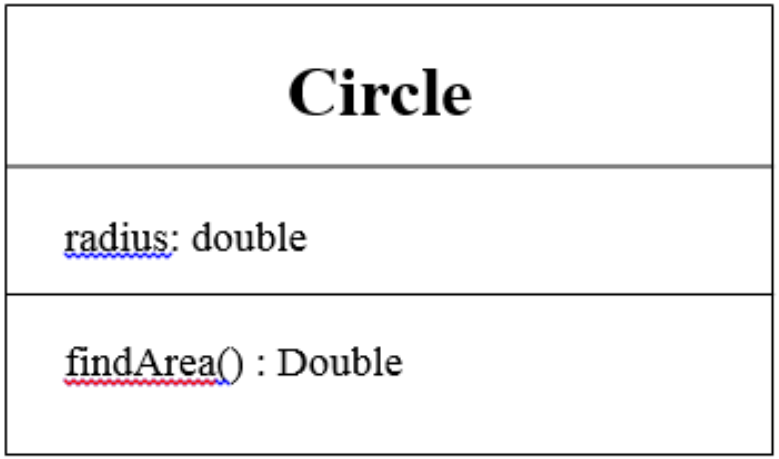
- Entered the mainstream in 1990's.
- Systems are modelled as a number of objects that are characterised by a number of operations and a state.
- An object represents an entity in the real world that can be distinctly identified.

Introduction



Defining Classes for Objects

- All objects have a unique identity, a state and behaviours.
- The state of an object consists of a set of fields, or fields with their current values.
- The behaviour of an object is defined by a set of methods.
- Consider the following UML class diagram:



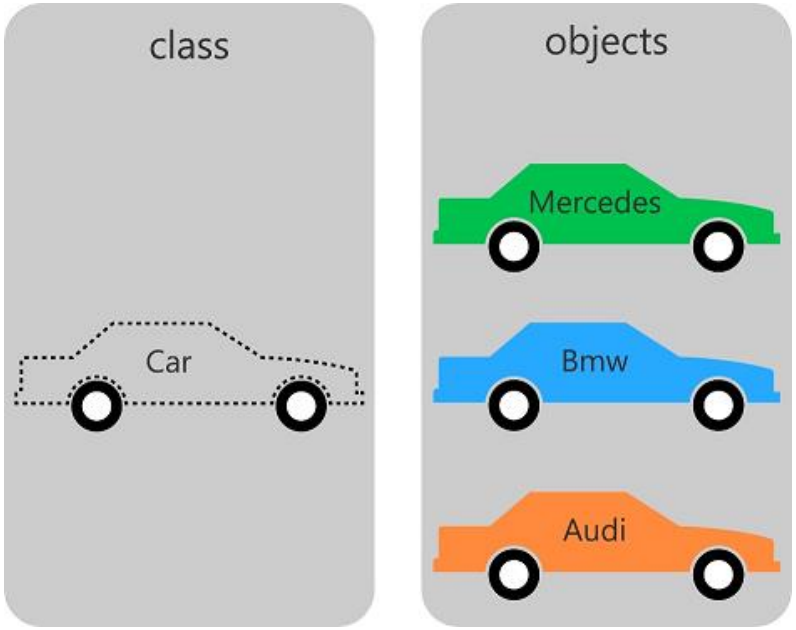
Defining Classes for Objects

Circle.java

```
1  class Circle {
2      double radius = 1.0;
3
4      double findArea() {
5          return radius*radius*3.14159;
6      }
7  }
```

Defining Classes for Objects

- This class does not have a `main` method and therefore you cannot run it.
- Classes are a definition used to create objects from.



Creating Objects

- Objects are created using the new operator as follows:

```
new className();
```

– `new Circle()` creates an object of the `Circle` class.

- Newly created objects are allocated memory and are accessed via object reference variables which contain references to the objects.

Creating Objects

- Such variables are declared using the following syntax.

```
ClassName objectReference;
```

- The types of reference variables are known as reference types.

```
Circle myCircle;
```

Creating Objects

- The variable myCircle can reference a Circle object.

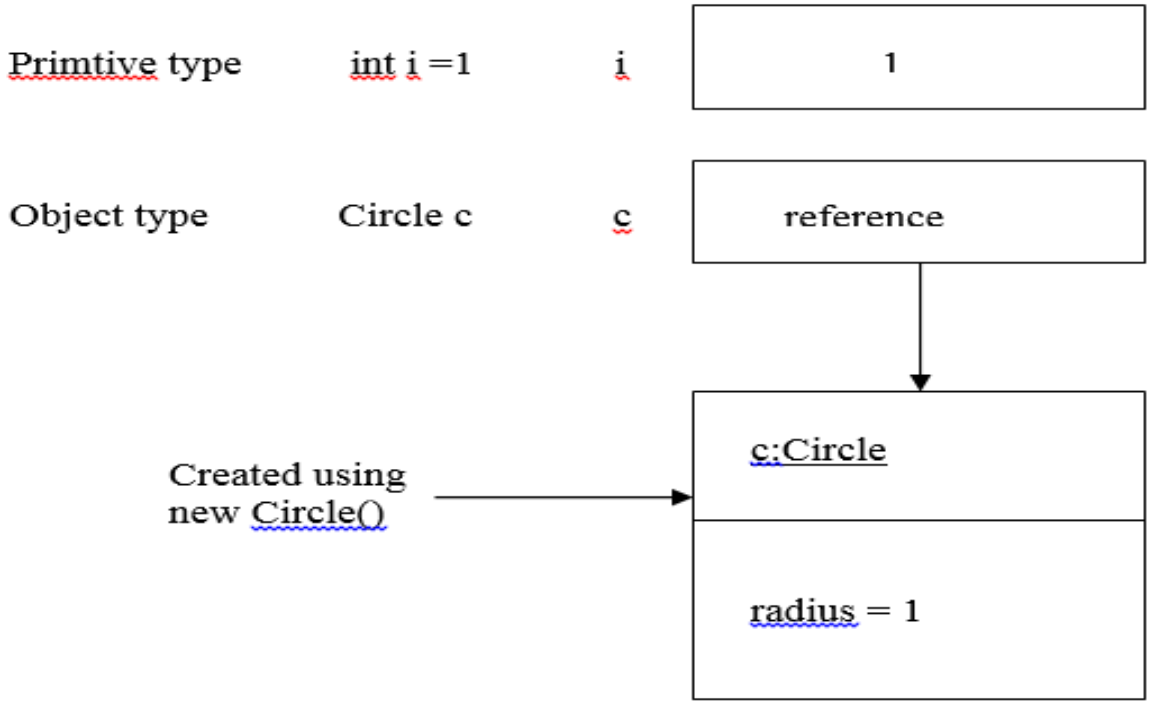
```
myCircle = new Circle();
```

- Often combine declaration and instantiation:

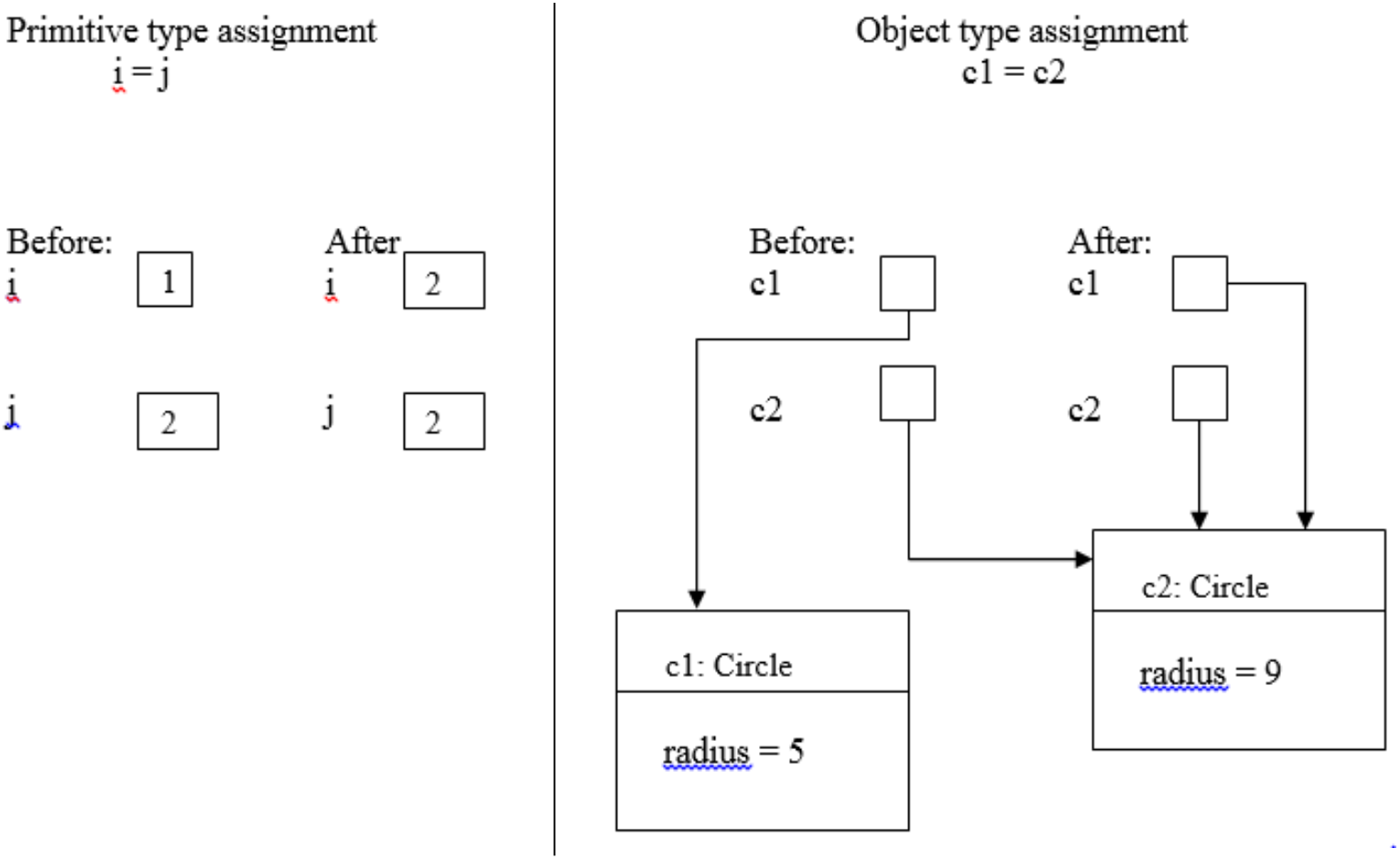
```
Circle myCircle = new Circle();
```

Differences Between Primitive and Reference Types

- Every variable represents a memory location that holds a value.
- For a primitive type, the value is of the primitive type.
- For a reference type, the value is a reference to where an object is located.



Differences Between Primitive and Reference Types



**Each object is independent of each other (and can have different values for its attributes)*

Garbage Collection

- After the assignment statement the object previously referenced by c1 is no longer useful.
- The object is considered to be garbage.
- Garbage occupies memory space.
- Once the JRE deems an object to be garbage it automatically reclaims the space it occupies.
- If you no longer require an object, assign it a null value and the garbage collector will clean up behind the scenes.

Accessing an Objects Data and Methods

- After an object has been created, its data can be accessed and its methods invoked using what is known as the dot notation.

```
objectReference.data

objectReference.method(arguments)
```

- For example:

```
myCircle.radius;

myCircle.findArea();
```

Accessing an Objects Data and Methods

- The data field *radius* is referred to as an instance variable because it is dependant on a specific instance.
- For the same reason, the method *findArea* is referred to as an instance method, because you can only invoke it on a specific (object) instance.

Accessing an Objects Data and Methods

```
3   class Circle {
4
5           double radius = 1.0;
6
7           double findArea() {
8                   return radius*radius*3.14159;
9           }//end findArea
10
11  }//end Circle
12
13  public class TestCircle {
14
15      public static void main(String args[]) {
16
17              Circle myCircle = new Circle();
18
19              System.out.println("The area of the circle of radius " +
20                                      myCircle.radius + " is " + myCircle.findArea());
21
22              System.exit(0);
23      }//end main
24  }//end TestCircle class
```

```
Output - OOSource (run)
run:
The area of the circle of radius 1.0 is 3.14159
BUILD SUCCESSFUL (total time: 1 second)
```

Constructors

- One problem with the Circle class is that all of the objects created from it have the same radius.
- It would be far more useful to create circles with radii of various lengths.
  - Can be achieved by the use of constructors.
- A constructor is a special method which can be used to initialise an objects data.
  - Can be used to assign an initial radius when you create an object.
  - Distinctive because they have the same name as the class.
- Like methods, constructors can be overloaded.

Constructors

- A constructor without any args is called a default constructor.
- It is worth noting the following about constructors:
  1. They must have the same name as the class itself.
  2. They do not have a return type – not even void.
  3. They are always invoked using the new operator when an object is created.
- .

Constructors

```
3   //define the Circle with two constructors
4   class Circle {
5           double radius;
6
7           //default constructor
8           Circle() {
9                   radius = 1.0;
10          }
11
12          //construct a circle with a specified radius
13          Circle(double r) {
14                  radius = r;
15          }
16
17          //return the area of this circle
18          double findArea() {
19                  return radius*radius*3.14159;
20          }
21  }//end Class Circle
```

Constructors

```
3   public class TestCircleWithConstructors {
4
5       public static void main(String[] args) {
6
7           Circle myCircle = new Circle(5.0);
8
9           System.out.println("The area of the circle of radius " +
10              myCircle.radius + " is " + myCircle.findArea());
11
12          Circle yourCircle = new Circle();
13
14          System.out.println("The area of the circle of radius " +
15              yourCircle.radius + " is " + yourCircle.findArea());
16
17          //modify circle radius
18          yourCircle.radius = 100;
19
20          System.out.println("The area of the circle of radius " +
21              yourCircle.radius + " is " + yourCircle.findArea());
22
23          System.exit(0);
24      }
25  }//end class CircleWithConstructors
```

Output - OOSource (run)

```
run:
The area of the circle of radius 5.0 is 78.53975
The area of the circle of radius 1.0 is 3.14159
The area of the circle of radius 100.0 is 31415.899999999998
BUILD SUCCESSFUL (total time: 0 seconds)
```

Visibility Modifiers & Accessor Methods

- The previous example works well, but it is not good practice to allow the client to modify values of instance variables directly through the object reference (line 18).
- To prevent direct modifications of the properties through the object reference, you can declare the field private.
- Java comes with three levels of visibility.
  - Public.
  - Private.
  - Protected.

Visibility Modifiers & Accessor Methods

- A mechanism is needed to access private data fields.
- You can provide a *get* method and a *set* (or mutator) method for a private data field.

- A **get** method has the following signature:

  public returnType getPropertyName()

- A **set** method has the following signature:

  public void setPropertyName(dataType propertyValue)

Visibility Modifiers & Accessor Methods

```
3   public class Circle {
4     private double radius;
5
6     /**Default constructor*/
7     public Circle()  {
8       radius = 1.0;
9     }//end default constructor
10
11    /**Construct a circle with a specified radius*/
      public Circle(double r) {
13      radius = r;
14    }//end constructor
15
16    /**Return radius*/
      public double getRadius() {
18      return radius;
19    }//end getRadius()
20
21    /**Set a new radius*/
      public void setRadius(double newRadius) {
23      radius = newRadius;
24    }//end setRadius(double)
25
26    /**Return the area of this circle*/
      public double findArea(){
28      return radius*radius*3.14159;
29    }//end findArea()
30  }
```

Visibility Modifiers & Accessor Methods

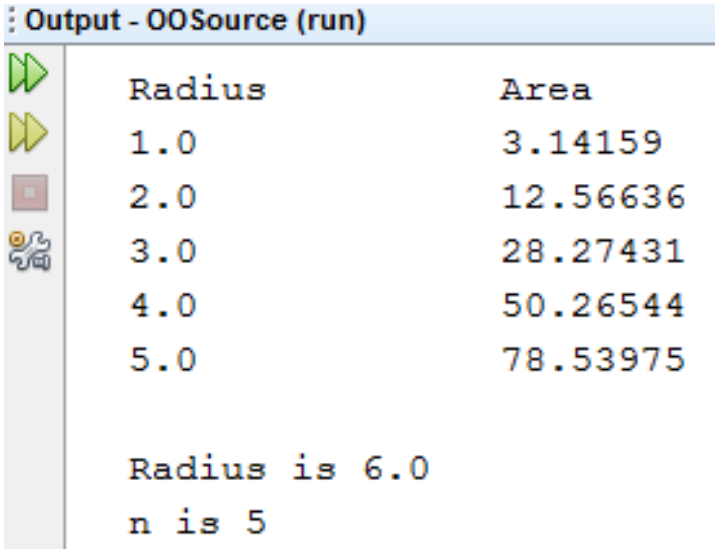
```
3   public class TestCircleWithAccessors {
4
5     /**Main method*/
6     public static void main(String[] args)    {
7
8       Circle myCircle = new Circle(5.0);
9       System.out.println("Radius: " + myCircle.getRadius() +
10                                      "\t Area: " + myCircle.findArea());
11
12      myCircle.setRadius(myCircle.getRadius()*1.1);
13      System.out.println("Radius: " + myCircle.getRadius() +
14                                      "\t Area: " + myCircle.findArea());
15
16      System.exit(0);
17
18    }
19  }
```

```
Output - OOSource (run)
run:
Radius: 5.0      Area: 78.53975
Radius: 5.5      Area: 95.0330975
```

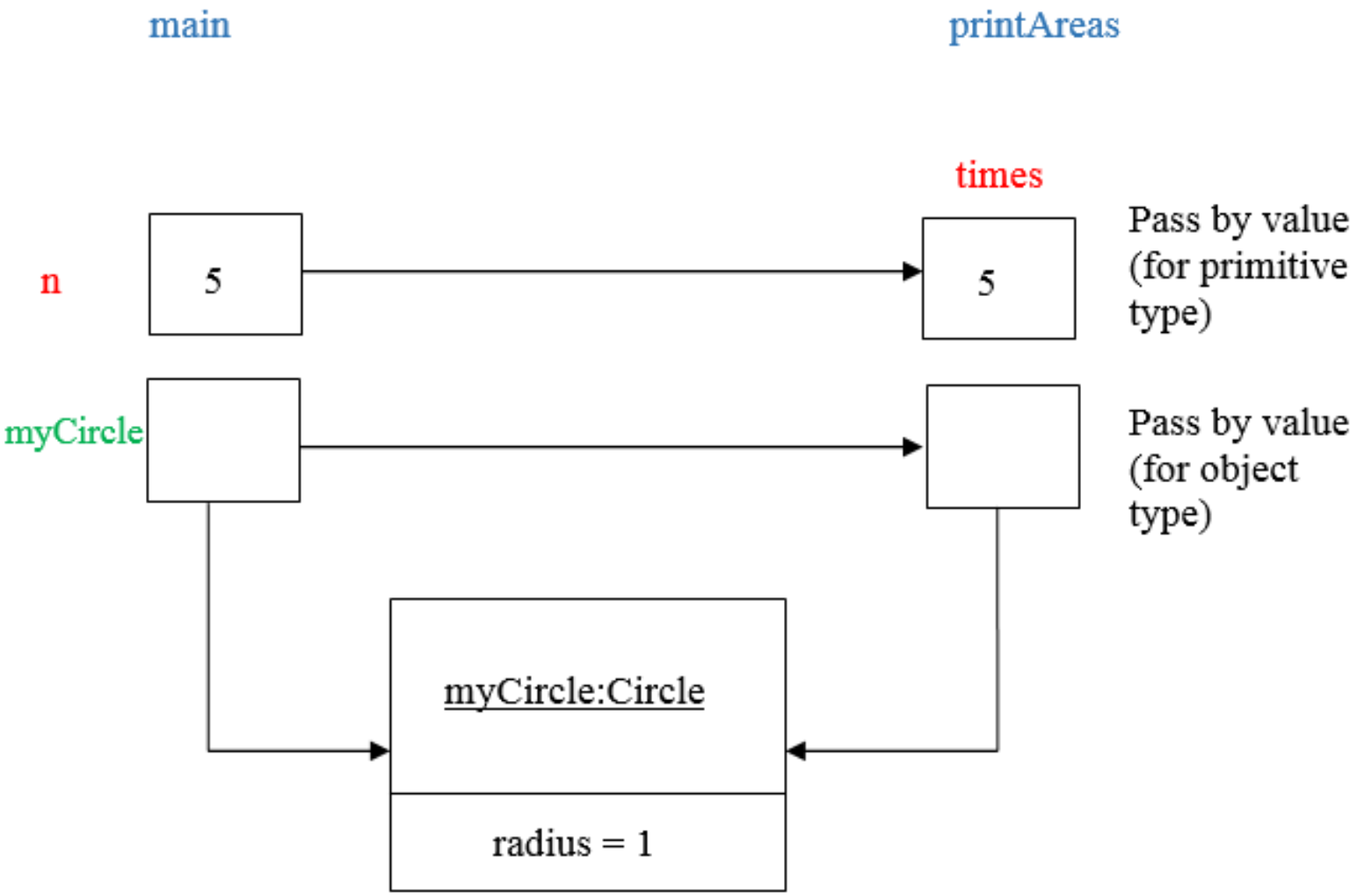
Passing Objects to Methods

```
7  public static void main(String[] args) {
8
9     // Create a Circle object with default radius 1
10    Circle myCircle = new Circle();
11
12    // Print areas for radius 1, 2, 3, 4, and 5.
13    int n = 5;
14    String finalOutput = printAreas(myCircle, n);
15
16    // See myCircle.radius and times
17    finalOutput += "\n" + "Radius is " + myCircle.getRadius();
18    finalOutput += "\nn is " + n;
19    System.out.println(finalOutput);
20    System.exit(0);
21  }
22  /**Print a table of areas for radius*/
    public static String printAreas(Circle c, int times) {
24   String output = "Radius \t\tArea \n";
25
26    while (times >= 1)  {
27
28      output += c.getRadius() + "\t\t" + c.findArea();
29      c.setRadius(c.getRadius()+1);
30      times--;
31      output +="\n";
32
33    }//end while
34
35    return output;
36
37  }//end printAreas
```

Passing Objects to Methods



Passing Objects to Methods



Static Variables, Constants & Methods

- The variable radius in the circle class is an instance variable.
  - Tied to a specific instance of the class.
  - Not shared among objects of the same class.
  - For example:

```
Circle circle1 = new Circle();
Circle circle2 = new Circle(5);
```

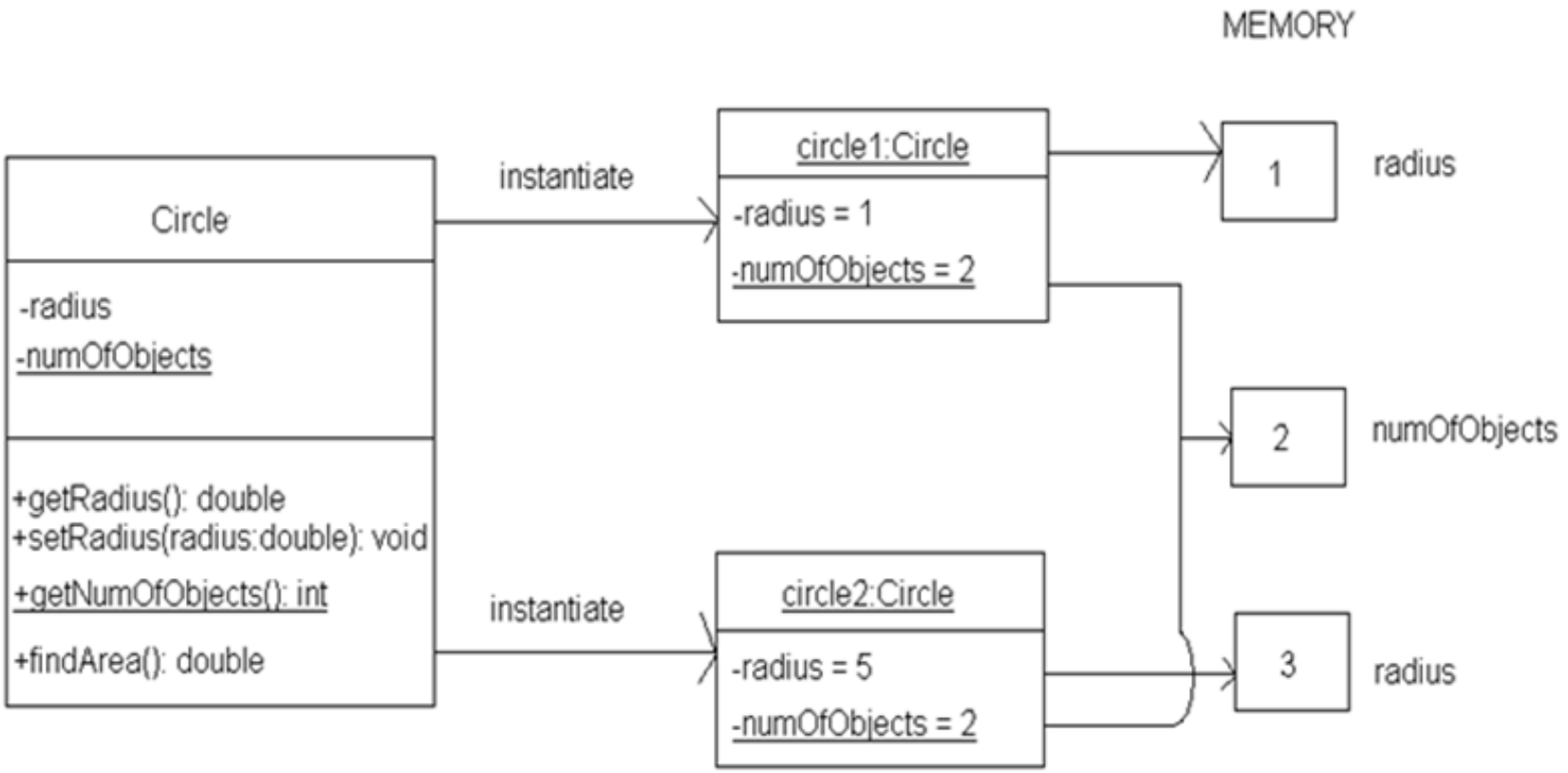
- The radius in circle1 is independent of the radius in circle2, and is stored in a different memory location.
  - Changes made to circle1's radius do not affect circle2's radius, and vice versa.

Static Variables, Constants & Methods

- If you want all the instance variables of a class to share data, use static variables.
- Static variables store values for the variables in a common memory location.
  - All objects of the same class are affected if one object changes the value of a static variable.
- To declare a static variable, put the modifier static in the variable declaration.

Static Variables, Constants & Methods

- Want to track the number of objects from a class that are created?
  - Use a static variable.



Static Variables, Constants & Methods

- To declare a class constant, add the *final* keyword in the preceding declaration.

```
public final static double PI = 3.14159;
```

Static Variables, Constants & Methods

- Java supports static methods as well as static variables.
- *Static methods*, (aka *class methods)*, can be called without creating an instance of the class.
- To define a static method, put the modifier *static* in the method declaration.

```
static returnValueType staticMethod();
```

- Static methods are called by one of the following syntaxes:

```
ClassName.methodName();
objectName.methodName();
```

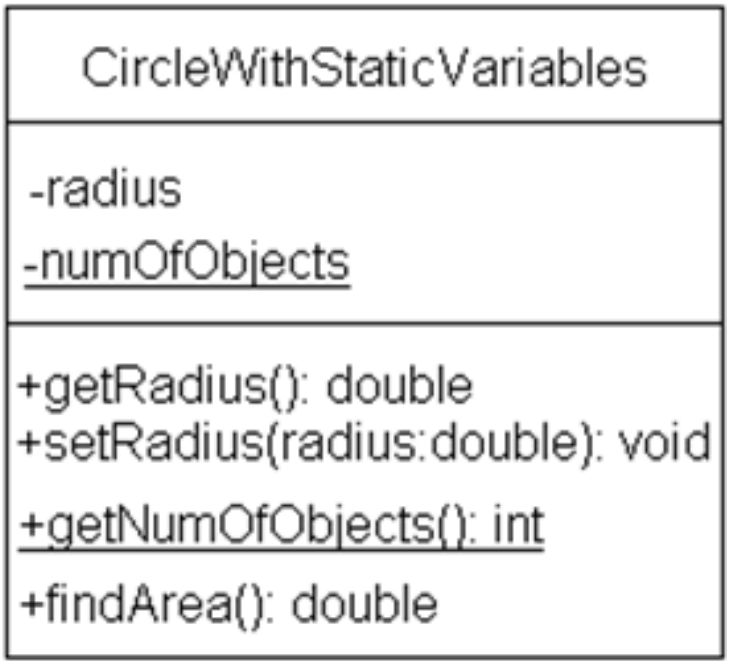
Static Variables, Constants & Methods

TIPS

- A method that does not use instance variables can be defined as a static method. This method can be invoked without creating an object of the class.
- You should define a constant as static data that can be shared by all class objects.
- Variables that describe common properties of objects should be declared as static variables.

Static Variables, Constants & Methods

- The following example shows how to use instance & static variables as well as methods, and illustrates the effects of using them. This example adds a static variable *numOfObjects* to track the number of circle objects created.



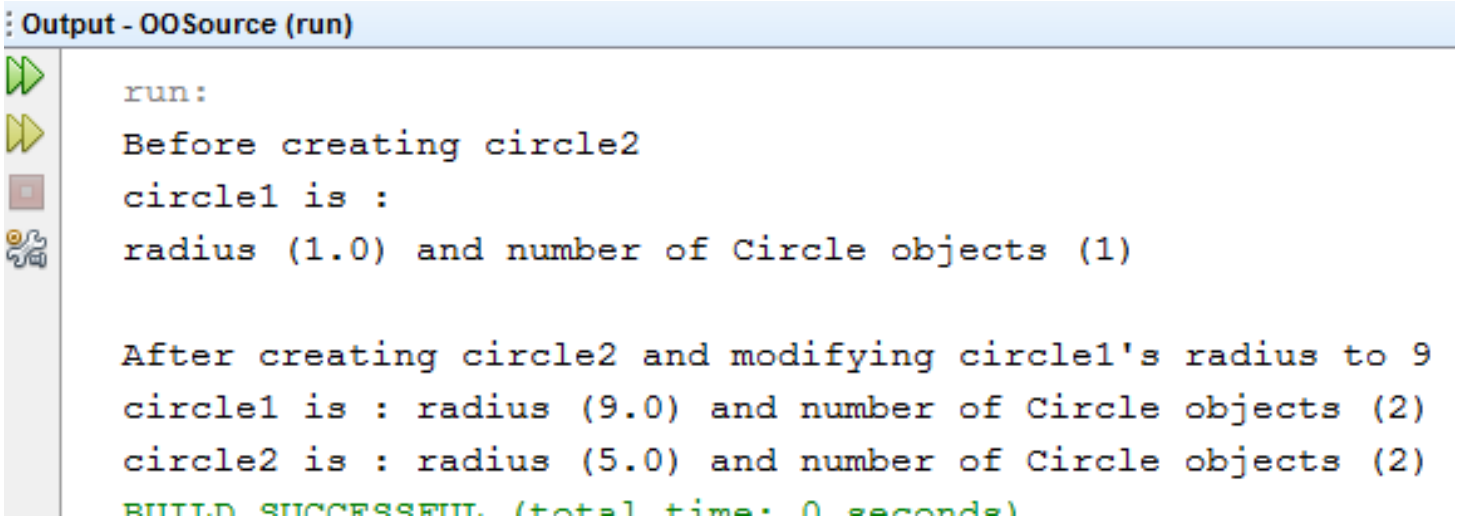
Static Variables, Constants & Methods

```
3   class Circle {
4     private double radius;
5     private static int numOfObjects = 0;   // Static variable
6
7     /**Default constructor*/
8     public Circle()   {
9       radius = 1.0;
10      numOfObjects++;
11    }
12
13    /**Construct a circle with a specified radius*/
14    public Circle(double r) {
15      radius = r;
16      numOfObjects++;
17    }
18
19    /**Return radius*/
20    public double getRadius() { return radius; }
21
22    /**Set a new radius*/
23    public void setRadius(double newRadius) { radius = newRadius; }
24
25    /**Return numOfObjects*/
26    public static int getNumOfObjects() { return numOfObjects; }
27
28    /**Return the area of this circle*/
29    public double findArea() {
30      return radius*radius*Math.PI;
31    }
32  }//end class Circle
```

Static Variables, Constants & Methods

```
public static void main(String[] args)    {
  // Create circle1
  Circle circle1 = new Circle();

  // Display circle1 BEFORE circle2 is created
  System.out.println("Before creating circle2\ncircle1 is : ");
  printCircle(circle1);

  // Create circle2
  Circle circle2 = new Circle(5);

  // Change the radius in circle1
  circle1.setRadius(9);

  // Display circle1 and circle2 AFTER circle2 was created
  System.out.println("\nAfter creating circle2 and modifying " +
    "circle1's radius to 9");
  System.out.print("circle1 is : ");
  printCircle(circle1);
  System.out.print("circle2 is : ");
  printCircle(circle2);
}//end main

/**Print circle information*/
public static void printCircle(Circle c)    {
  System.out.println("radius (" + c.getRadius() +
    ") and number of Circle objects (" +
    c.getNumOfObjects() + ")");
}//end printCircle
```

Static Variables, Constants & Methods



Static Imports

- Since Java 5.0 there is a variant of the standard import statement that lets you use static methods and variables without using class prefixes.

```
1  import static java.lang.System.*;
2  import static java.lang.Math.*;
3
4  class RootTester {
5
6      public static void main(String[] args) {
7          double r = sqrt(PI);  // instead of:  double r = Math.sqrt(Math.PI);
8          out.println(r);       // instead of: System.out.println(r);
9
10     }//end main
11
12 }//end class
```

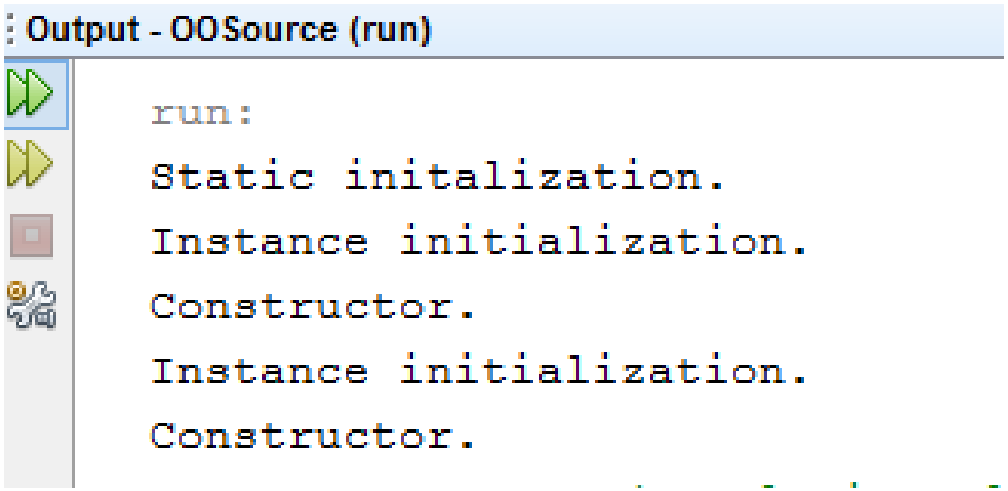
Initialization Blocks

- Instance variables are initialized with a default value (0, false, 0.0, etc) which depends on their type.
- You can set them to any value in a constructor and that is the desired approach.
- Another (often rarely used) mechanism exists whereby you can place one or more initialization block inside the class declaration.
  - All statements in that block are executed whenever an object is being constructed.
- In fact, there are two types of initialization blocks – an instance initialization block and a static initialization block.

Initialization Blocks

```
1   public class Test {
2
3       static int staticVariable;
4       int instanceVariable;
5
6       // Static initialization block:
7       // Runs once (when the class is initialized).
8       static {
9           System.out.println("Static initalization.");
10          staticVariable = 5;
11      }
12
13      // Instance initialization block:
14      // Runs before the constructor each time you instantiate an object
15      {
16          System.out.println("Instance initialization.");
17          instanceVariable = 7;
18      }
19
20      public Test() {
21          System.out.println("Constructor.");
22      }//end Test()
23
24      public static void main(String[] args) {
25          Test test1 = new Test();
26          Test test2 = new Test();
27      }//end main
28
29  } //end class
```

Initialization Blocks



The Scope of Variables

- Local variables are declared and used inside a method locally.
- The scope of a class's variables (instance and static variables) is the entire class.
- A class's variables can be declared anywhere in the class.
- Variable's can be declared at the end of a class and used it in a method defined earlier in the class…

```
3   class Circle {
4           double findArea() {
5                   return radius*radius*Math.PI;
6           }
7           double radius = 1;
8   }
9
```

The Scope of Variables

- The data fields and methods are the members of the class.
- There is no order among them.
- Therefore, they can be declared in any order in a class.
- The exception to this is when a data field is initialised based on a reference of another data field.
- The other data field must not be declared before this data.
- In the following example, `i` must be declared before `j`, because `i`'s value is used to initialise `j`.

The Scope of Variables

- You can declare a variable only once as a class member (instance variable or static variable), but you can declare the same variable in a method multiple times in different non-nesting blocks.
- If a local variable has the same name as a class's variable, the local variable takes precedence and the class's variable with the same name is hidden.

```
1   class MyClass {
2
3       int x = 0; //instance variable
4       int y = 0;
5
6       MyClass() {
7       }
8
9       void p() {
            int x = 1; //local variable
11          System.out.println("x = " + x);
12          System.out.println("y = " + y);
13
14      }
15  }//end MyClass
```

The Keyword *this*

- Within an instance method or a constructor, *this* is a reference to the current object.
- The object whose method or constructor is being called.
- You can refer to any member of the current object from within an instance method or a constructor by using *this*.

```
1   class MyClass {
2
3       int i = 5;
4
5       void setI(int i) {
6           this.i = i;  //assign argument i to
7                        //the objects's data field i
8       }
9   }
```

The Keyword *this*

- Can also be used to call constructors:

```
2   public class Circle {
3
4       private double radius;
5
6       public Circle(double radius) {
7           this.radius = radius;
8       }
9
10      public Circle() {
11          this(1.0);
12      }
13
14      public double findArea() {
15          return radius * radius * Math.PI;
16      }
17
18  }
```

The Keyword *this*

- `this` always refers to the currently executing object.
  - In a class called `ChessPiece` there could be a method `move`, which might contain the following line.

```
if (this.position == piece2.position)
        result = false;
```

- Here `this` is being used to clarify which position is being referenced.
- `this` refers to the object through which the method was invoked.

The Keyword *this*

- When the following line is used to invoke the method, the `this` reference refers to bishop1.

```
bishop1.move();
```

- When another object is used to invoke the method, `this` refers to it. Therefore, when the following invocation is used, the `this` reference in the move method refers to bishop2.

```
bishop2.move();
```

Arrays of Objects

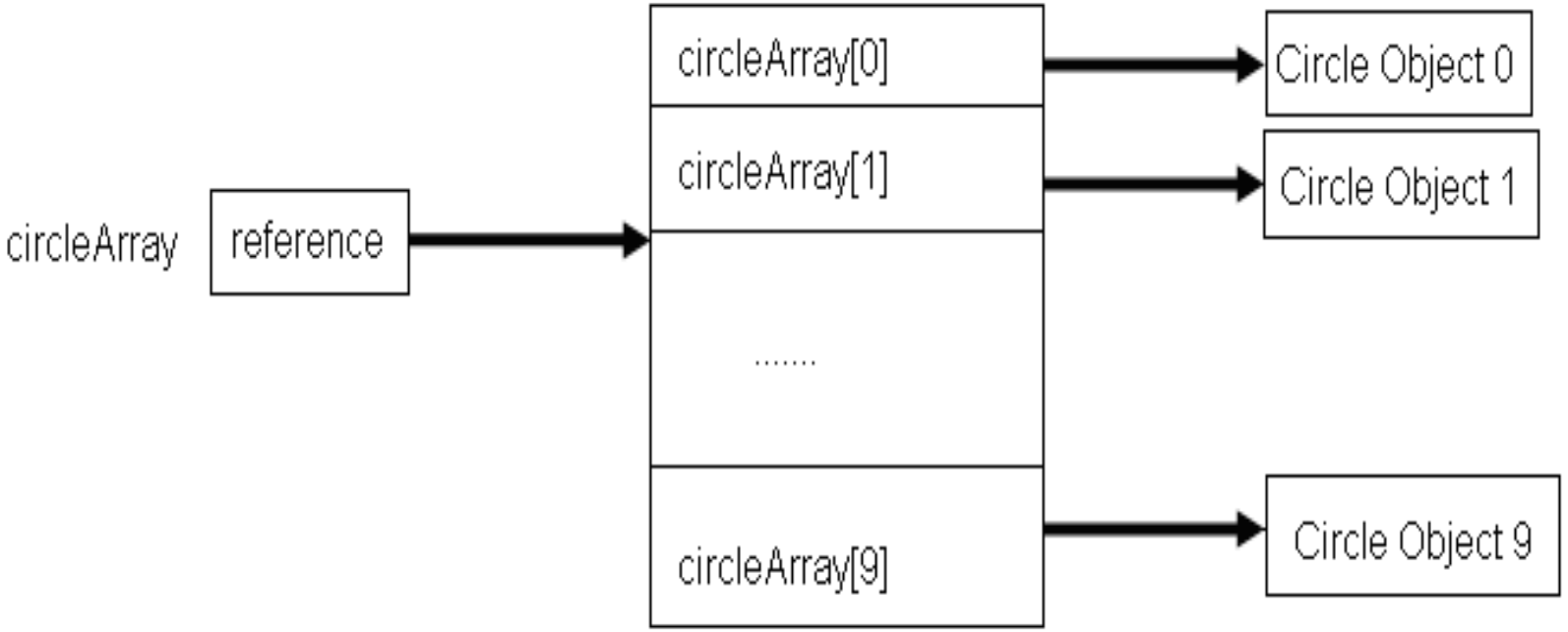
- Arrays can be used to create arrays of objects. The following declares and creates an array of 10 *Circle* objects:

```
Circle[] carr = new Circle[10];
```

- To initialise the circleArray you might use code like this:

```
for (int = 0; i<carr.length; i++)
      circleArray[1] = new Circle();
```

Arrays of Objects



ArrayLists of Objects

- `ArrayLists` are frequently used when storing collections of objects.

```
9           ArrayList<Circle> aList = new ArrayList();
10
11          for (int i = 1; i <=10; i++) {
12              aList.add(new Circle(i));
13          }
14
15          for (Circle aCircle : aList )
16              System.out.println(aCircle.findArea());
```

```
Output - OOSource (run)
run:
3.141592653589793
12.566370614359172
28.274333882308138
50.26548245743669
78.53981633974483
113.09733552923255
153.93804002589985
201.06192982974676
254.46900494077323
314.1592653589793
```

References

https://history-computer.com/ModernComputer/Software/Simula.html

https://docs.sencha.com/extjs/6.0.2-classic/guides/other_resources/oop_concepts.html

Paul J Deitel (2016) *Java How To Program.* 10/E. ISBN-13 9780134800271 (Link)

Y. Daniel Liang (2014) *Intro to Java Programming and Data Structures, Comprehensive Version.* 11/E. Pearson. ISBN-13 978-0134670942 (Link)