Constructors

Exercises and solutions

1. What is a constructor? What is the name of the operator that must be used along with a constructor to create an object of a class?

**Answer:**

A constructor is a named block of code that is used to initialize an object of a class immediately after the object is created. The new operator is used to create an object of a class.

1. What is a default constructor? What is the access level of a default constructor?

**Answer:**

The constructor that is added by the compiler is called the default constructor. The default constructor is added when you do not declare any constructor for a class. The default constructor takes no parameters. The access level of the default constructor is the same as the access level of the class

1. How do you call a constructor of a class from another constructor of the same class? Describe any restrictions where such a call should be placed in the code.

**Answer:**

Use the keyword this to call a constructor from another constructor. There are two rules:

* The call to another constructor must be the first statement in the constructor.
* A constructor cannot call itself.

1. What are static and instance initializers?

**Answer:**

An instance initialization block, also called instance initializer, is a block of code inside the body of a class, but outside any methods or constructors. Like constructors, it is used to initialize objects of a class.

A static initialization block, also called a static initializer, is a block of code inside the body of a class, but outside any methods or constructors. It is used to initialize class variables.

1. What are final variables and blank final variables?

**Answer:**

Variable whose value cannot be modified once it has been initialized is called final variable. A final variable, which is not initialized at the time of its declaration is known as blank final variable. A blank final variable must be initialized once and only once before it can be read.

1. What is the effect of declaring a method's parameter or a constructor's parameter final?

**Answer:**

The final parameter cannot be changed inside method’s or constructor’s body.

1. Consider the following code for a Cat class:   
     
   // Cat.java  
   package com.jdojo.cls.excercise;  
     
   public class Cat {  
   }  
     
   When the Cat class is compiled, the compiler will add a default constructor to it. Rewrite the Cat class as if you are adding the default constructor instead of the compiler.

**Solution:**

public class Cat {

**public Cat() {**

**}**  
}

1. Consider the following code for a Mouse class:   
     
   // Mouse.java  
   package com.jdojo.cls.excercise;  
     
   class Mouse {  
   }  
     
   When the Mouse class is compiled, the compiler will add a default constructor to it. Rewrite the Mouse class as if you are adding the default constructor instead of the compiler.

**Solution:**

// Mouse.java

class Mouse {

**Mouse() {**

**}**  
}

1. Create a SmartPoint2D class with two int instance variables named x and y. The instance variables should be declared private and final. An instance of the SmartPoint2D class represents an immutable point in a 2-D plane. That is, once an object of the SmartPoint2D class is created, the x and y values of that object cannot be changed. Add a public constructor to the class, which should accept the values for the two instance variables x and y and initialize them with the passed in values.

**Solution:**

// SmartPoint2D.java

package com.jdojo.cls.excercise;

public class SmartPoint2D {

private final int x;

private final int y;

public SmartPoint2D(int x, int y) {

this.x = x;

this.y = y;

}

}

1. Add getters for the x and y instance variables in the SmartPoint2D class that you created in the previous exercise.

**Solution:**

// SmartPoint2D.java

package com.jdojo.cls.excercise;

public class SmartPoint2D {

private final int x;

private final int y;

public SmartPoint2D(int x, int y) {

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

}

1. Add a public static final variable named ORIGIN to the SmartPoint2D class. The ORIGIN variable is of the SmartPoint2D class and it is a SmartPoint2D with x = 0 and y = 0.

**Solution:**

// SmartPoint2D.java

package com.jdojo.cls.excercise;

public class SmartPoint2D {

private final int x;

private final int y;

public static SmartPoint2D ORIGIN = new SmartPoint2D(0, 0);

public SmartPoint2D(int x, int y) {

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

}

1. Implement a method named distance in the SmartPoint2D class that you created in the previous exercise. The method accepts an instance of the SmartPoint2D class and returns the distance between the current point and the point represented by the parameter. The method should be declared as follows:  
     
   public class SmartPoint2D {  
    /\* Code from the previous exercise goes here. \*/  
     
    public double distance(SmartPoint2D p) {  
    /\* Your code for this exercise goes here. \*/  
    }  
   }  
     
   **Hint**: The distance between two points (x1, y1) and (x2, y2) is computer as . You can use Math.sqrt(n) method to compute the square root of a number n.

**Solution:**

// SmartPoint2D.java

package com.jdojo.cls.excercise;

public class SmartPoint2D {

private final int x;

private final int y;

public static SmartPoint2D ORIGIN = new SmartPoint2D(0, 0);

public SmartPoint2D(int x, int y) {

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public double distance(SmartPoint2D p) {  
 int diffX = this.x - p.x;

int diffY = this.y - p.y;

return Math.sqrt(diffX \* diffX + diffY \* diffY);

}

}

1. Create a Circle class, which has three private final instance variables named x, y, and radius. The x and y instance variables represent the x and y coordinates of the center of the circle; they are of int data type. The radius instance variable represents the radius of the circle; it is of the double data type. Add a constructor to the Circle class that accepts the values for its instance variables x, y, and radius. Add getters for the three instance variables.

**Solution:**

// Circle.java

package com.jdojo.cls.excercise;

public class Circle {

private final int x;

private final int y;

private final double radius;

public Circle(int x, int y, double radius) {

this.x = x;

this.y = y;

this.radius = radius;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public double getRadius() {

return radius;

}

}

1. Enhance the Circle class by adding four instance methods named centerDistance, distance, overlaps, and touches. All these methods accept a Circle as a parameter. The centerDistance method returns the distance (as a double) between the centers of the circle and another circle passed in as the parameter. The distance method returns the minimum distance (as a double) between the two circles. If two circles overlap, the distance method returns a negative number. The overlaps method returns true if two circles overlaps, false otherwise. The touches method returns true if two circles touches each other; false otherwise. The distance method must make use of the centerDistance method. The body of the overlaps and touches methods must contain only one statement that makes use of the distance method.  
   Hint: Distance between two circles is the distance between their centers minus their radii. Two circle overlaps if the distance between them is negative. Two circles touch if the distance between them is zero.

**Solution:**

// Circle.java

package com.jdojo.cls.excercise;

public class Circle {

private final int x;

private final int y;

private final double radius;

public Circle(int x, int y, double radius) {

this.x = x;

this.y = y;

this.radius = radius;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public double getRadius() {

return radius;

}

public double centerDistance (Circle c) {  
 int diffX = this.x – c.getX();

int diffY = this.y – c.getY();

return Math.sqrt(diffX \* diffX + diffY \* diffY);

}

public double distance(Circle c) {  
 return this.centerDistance(c) – this.radius – c.getRadius();

}

public boolean overlaps(Circle c) {

return this.distance(c) < 0;

}

public boolean touches(Circle c) {

return this.distance(c) == 0;

}

}

1. Enhance the Circle class by adding two methods named perimeter and area that compute and return the perimeter and area of the circle, respectively.

**Solution:**

// Circle.java

package com.jdojo.cls.excercise;

public class Circle {

private final int x;

private final int y;

private final double radius;

public Circle(int x, int y, double radius) {

this.x = x;

this.y = y;

this.radius = radius;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public double getRadius() {

return radius;

}

public double perimeter() {  
 return 2 \* Math.PI \* radius;

}

public double area() {  
 return Math.PI \* radius \* radius;

}

public double centerDistance(Circle c) {  
 int diffX = this.x – c.getX();

int diffY = this.y – c.getY();

return Math.sqrt(diffX \* diffX + diffY \* diffY);

}

public double distance(Circle c) {  
 return this.centerDistance(c) – this.radius – c.getRadius();

}

public boolean overlaps(Circle c) {

return this.distance(c) < 0;

}

public boolean touches(Circle c) {

return this.distance(c) == 0;

}

}

1. Add a second constructor to the Circle class that takes a double parameter, which is the radius of the circle. This constructor should call another existing constructor of the Circle class with three parameters passing zero as the values for x and y.

**Solution:**

// Circle.java

package com.jdojo.cls.excercise;

public class Circle {

private final int x;

private final int y;

private final double radius;

public Circle(int x, int y, double radius) {

this.x = x;

this.y = y;

this.radius = radius;

}

public Circle(double radius) {

this(0, 0, radius);

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public double getRadius() {

return radius;

}

public double perimeter() {  
 return 2 \* Math.PI \* radius;

}

public double area() {  
 return Math.PI \* radius \* radius;

}

public double centerDistance(Circle c) {  
 int diffX = this.x – c.getX();

int diffY = this.y – c.getY();

return Math.sqrt(diffX \* diffX + diffY \* diffY);

}

public double distance(Circle c) {  
 return this.centerDistance(c) – this.radius – c.getRadius();

}

public boolean overlaps(Circle c) {

return this.distance(c) < 0;

}

public boolean touches(Circle c) {

return this.distance(c) == 0;

}

}

1. A double value can be NaN, positive infinity, and negative infinity. Enhance the constructor of the Circle class with three parameters, x, y, and radius, so it throws a RuntimeException when the value of the radius parameter is not a finite number or a negative number.  
     
   Hint: The java.lang.Double class contains a static isFinite(double n) method, which returns true if the specified parameter n is a finite number; false otherwise. Use the following statement to throw a RuntimeException:  
     
   throw new RuntimeException("Radius must be a finite non-negative number.");

**Solution:**

This if the final version of the Circle class with comments and a main() method to test the class.

// Circle.java

package com.jdojo.cls.excercise;

public class Circle {

private final int x;

private final int y;

private final double radius;

public Circle(double radius) {

this(0, 0, radius);

}

public Circle (int x, int y, double radius) {

if(!Double.isFinite(radius) || radius < 0.0) {

throw new RuntimeException("Radius must be a finite non-negative number.");

}

this.x = x;

this.y = y;

this.radius = radius;

}

public static void main(String[] args) {

//new Circle(0, 0, -10);

Circle c1 = new Circle(0, 0, 10);

Circle c2 = new Circle(15, 0, 5);

double originDistance = c1.centerDistance(c2);

double distance = c1.distnace(c2);

boolean overlaps = c1.overlaps(c2);

boolean touches = c1.touches(c2);

System.out.println("Circle 1 perimeter = " + c1.perimeter());

System.out.println("Circle 1 area = " + c1.perimeter());

System.out.println("Circle 2 perimeter = " + c2.perimeter());

System.out.println("Circle 2 area = " + c2.area());

System.out.println("Origin distance = " + originDistance);

System.out.println("Distance = " + distance);

System.out.println("Overlaps = " + overlaps);

System.out.println("Touches = " + touches);

}

public double perimeter() {

return 2 \* Math.PI \* getRadius();

}

public double area() {

return Math.PI \* getRadius() \* getRadius();

}

public double centerDistance(Circle c) {

int diffX = this.getX() - c.getX();

int diffY = this.getY() - c.getY();

double dist = Math.sqrt(diffX \* diffX + diffY \* diffY);

return dist;

}

public double distnace(Circle c) {

// Get the distance between the origin of two circles

double centerDistance = this.centerDistance(c);

// Subtract the radius of the two circles to get the distance between two circles

double dist = centerDistance - this.getRadius() - c.getRadius();

return dist;

}

public boolean overlaps(Circle c) {

// If the distance between two circles is negative, it means they overlap

return this.distnace(c) < 0;

}

public boolean touches(Circle c) {

// If the distance between two circles is zero, it means they touch each other.

return this.distnace(c) == 0;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public double getRadius() {

return radius;

}

}

1. Consider the following InitializerTest class? How many static and instance initializer are in this class? What will be printed when this class is run?  
     
   // InitializerTest.java  
   package com.jdojo.cls.excercise;  
     
   public class InitializerTest {  
    private static int count;  
     
    {  
    System.out.println(count++);  
    }  
     
    {  
    System.out.println(count++);  
    }  
     
    static {  
    System.out.println(count);  
    }  
     
    public static void main(String[] args) {  
    new InitializerTest();  
    new InitializerTest();  
    }  
   }

**Answer:**

There are two instance initializers and one static initializer in the InitializerTest class. The following is the output when the class is run:

0

0

1

2

3

Here are the explanations for each of the five lines in the output:

* When the class is loaded, the count static variable is initialized to zero and the static initializer prints the current value of the count variable, which is zero,
* When you create the first object of the class, the first instance initializer prints the current value of the count variable, which is zero, and increments its value to 1. After the first instance initializer is finished, the current value of the count variable is 1.
* When you create the first object of the class, the second instance initializer prints the current value of the count variable, which is 1, and increments its value to 2. After the second instance initializer is finished, the current value of the count variable is 2.
* When you create the second object of the class, the first instance initializer prints the current value of the count variable, which is 2, and increments its value to 3. After the first instance initializer is finished, the current value of the count variable is 3.
* When you create the second object of the class, the second instance initializer prints the current value of the count variable, which is 3, and increments its value to 4. After the second instance initializer is finished, the current value of the count variable is 4.

1. Describe the reason why the following FinalTest class does not compile.   
     
   // FinalTest.java  
   package com.jdojo.cls.excercise;  
     
   public class FinalTest {  
    public static int square(final int x) {  
    x = x \* x;   
    return x;  
    }  
   }

**Answer:**

The final parameter x cannot be assigned a new value inside the square() method.

1. Describe the reason why the following BlankFinalTest class does not compile.  
     
   // BlankFinalTest.java  
   package com.jdojo.cls.excercise;  
     
   public class BlankFinalTest {  
    private final int x;  
    private final int y;  
     
    {  
    y = 100;  
    }  
     
    public BlankFinalTest() {  
    y = 100;  
    }  
     
    /\* More code goes here \*/  
   }

**Answer:**

The blank final variable y can be assigned a value only once, either in the instance initializer or in the constructor, but not in both.