



# Programming Fundamentals 1

---

Produced  
by  
Mr. Dave Drohan ([david.drohan@setu.ie](mailto:david.drohan@setu.ie))  
Dr. Siobhán Drohan  
Ms. Mairead Meagher

Department of Computing & Mathematics  
South East Technological University  
Waterford, Ireland

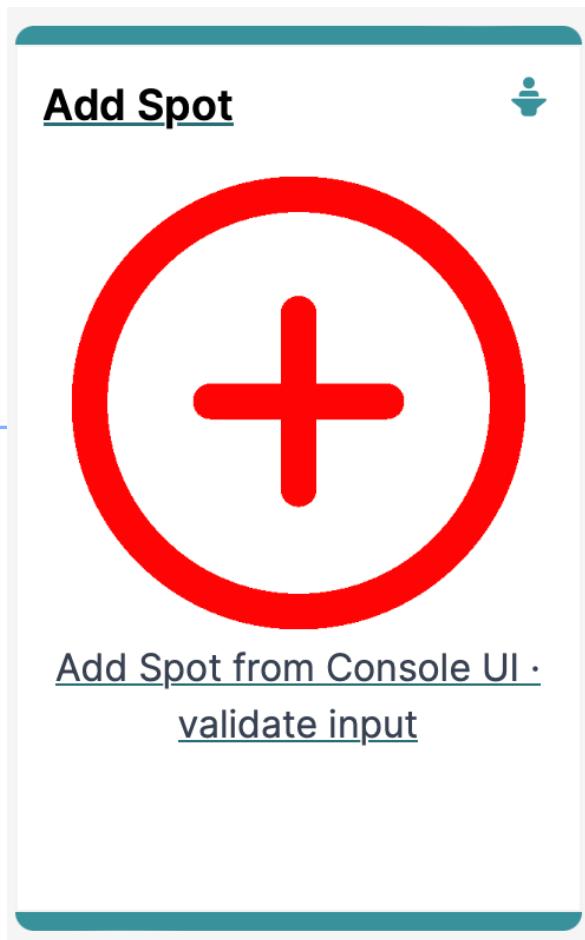
setu.ie





# IntelliJ and Spot

Add Spot (and Validate Input)





# Agenda

---

- ❑ RECAP
- ❑ Spot Constructors
- ❑ Add a Spot
- ❑ User Input Validation

---

# RECAP





# RECAP: Scanner class

```
import java.util.Scanner;  
  
public class Driver {  
  
    Spot spot = new Spot();  
    Scanner input = new Scanner(System.in);
```

Driver class

Scanner Class to  
read from the console





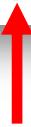
# RECAP: Driver class

```
import java.util.Scanner;  
  
public class Driver {  
  
    Spot spot = new Spot();  
    Scanner input = new Scanner(System.in);
```

Driver class

Method to ask the user to enter new values for the three fields.

```
void updateSpotDetails(){  
    System.out.print("Enter new xCoord value: ");  
    float enteredXCoord = input.nextFloat();  
    System.out.print("Enter new yCoord value: ");  
    float enteredYCoord = input.nextFloat();  
    System.out.print("Enter new diameter value: ");  
    float enteredDiameter = input.nextFloat();  
    spot.setxCoord(enteredXCoord);  
    spot.setyCoord(enteredYCoord);  
    spot.setDiameter(enteredDiameter);  
}
```



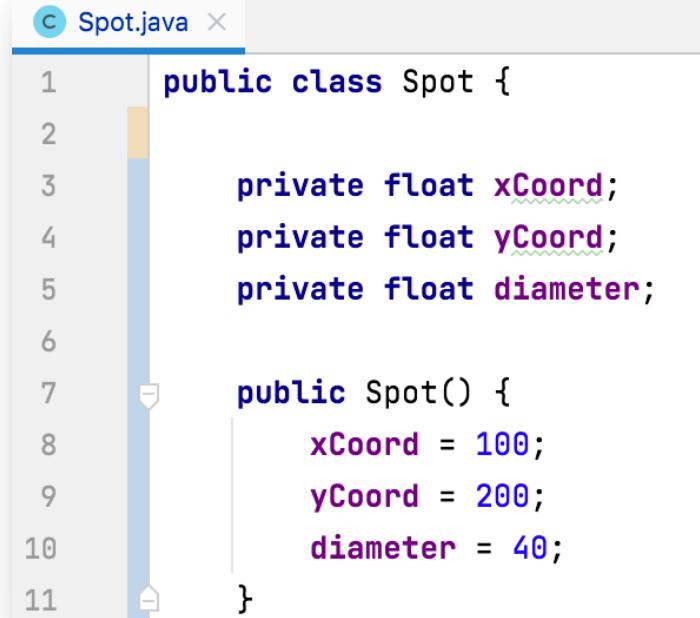
---

# Spot Constructors



# Spot Constructor

- ❑ Currently, the **Spot** class only has one constructor
- ❑ It is the **default** constructor as it has no parameters
- ❑ This constructor sets default values for each field



```
c Spot.java ×  
1 public class Spot {  
2  
3     private float xCoord;  
4     private float yCoord;  
5     private float diameter;  
6  
7     public Spot() {  
8         xCoord = 100;  
9         yCoord = 200;  
10        diameter = 40;  
11    }  
12}
```

A screenshot of a Java code editor showing the Spot.java file. The code defines a class Spot with three private float fields: xCoord, yCoord, and diameter. It contains a single-parameter constructor that initializes these fields to 100, 200, and 40 respectively.

# Spot Constructor – adding a second one

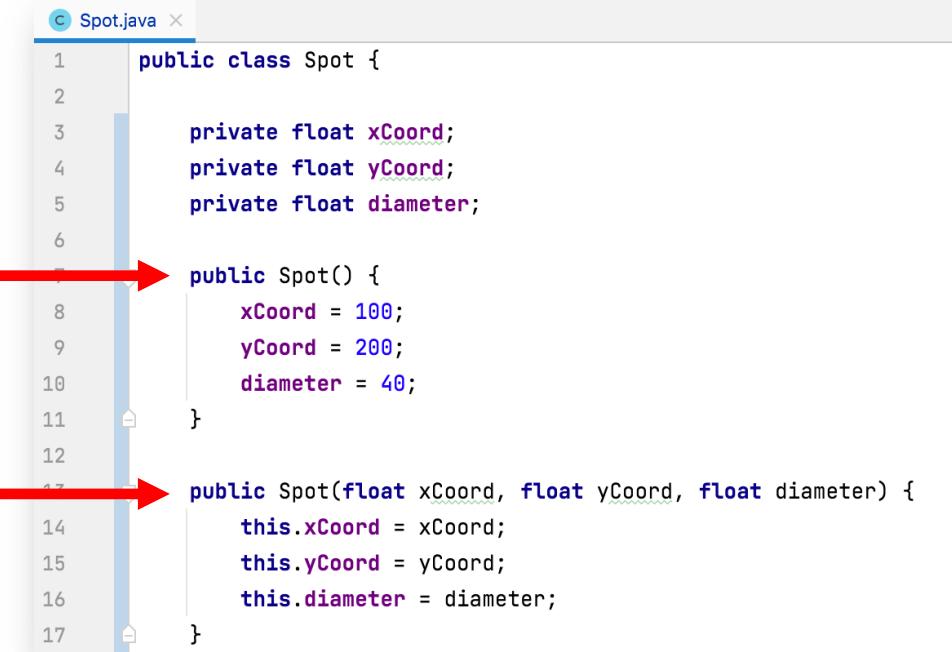
- ☐ If we want to allow the user to add a new Spot with their own data (read from the console), then we need to provide a second constructor that will allow this



```
Spot.java
1 public class Spot {
2
3     private float xCoord;
4     private float yCoord;
5     private float diameter;
6
7     public Spot() {
8         xCoord = 100;
9         yCoord = 200;
10        diameter = 40;
11    }
12
13    public Spot(float xCoord, float yCoord, float diameter) {
14        this.xCoord = xCoord;
15        this.yCoord = yCoord;
16        this.diameter = diameter;
17    }
}
```

# Spot Constructor – overloading

- We now have two constructors, both called **Spot**, with different parameter lists in our class
- This is called **constructor overloading**



```
Spot.java
public class Spot {

    private float xCoord;
    private float yCoord;
    private float diameter;

    public Spot() {
        xCoord = 100;
        yCoord = 200;
        diameter = 40;
    }

    public Spot(float xCoord, float yCoord, float diameter) {
        this.xCoord = xCoord;
        this.yCoord = yCoord;
        this.diameter = diameter;
    }
}
```



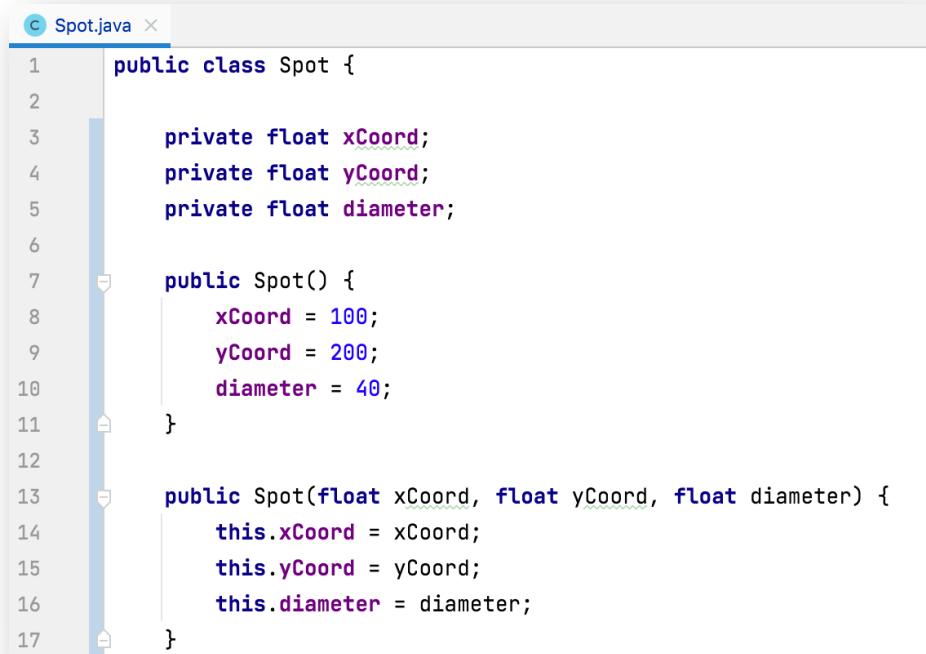
# Spot Constructor – overloading

- Why would we need two different constructors?

```
Spot.java x
1  public class Spot {
2
3      private float xCoord;
4      private float yCoord;
5      private float diameter;
6
7      public Spot() {
8          xCoord = 100;
9          yCoord = 200;
10         diameter = 40;
11     }
12
13     public Spot(float xCoord, float yCoord, float diameter) {
14         this.xCoord = xCoord;
15         this.yCoord = yCoord;
16         this.diameter = diameter;
17     }
}
```

# Spot Constructor – overloading

- Why would we need two different constructors?
- It gives us different options for creating a **Spot** object:
  - When we have no values for the fields
  - When we have values for the fields (from the user, maybe)



```
Spot.java
public class Spot {

    private float xCoord;
    private float yCoord;
    private float diameter;

    public Spot() {
        xCoord = 100;
        yCoord = 200;
        diameter = 40;
    }

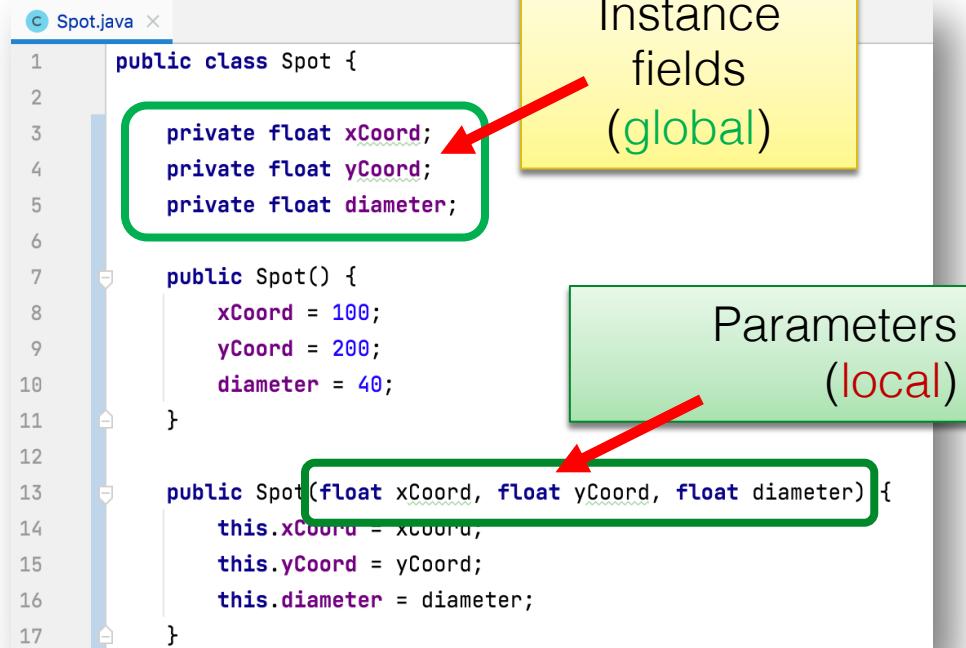
    public Spot(float xCoord, float yCoord, float diameter) {
        this.xCoord = xCoord;
        this.yCoord = yCoord;
        this.diameter = diameter;
    }
}
```

# Spot Constructor – `this` keyword

Both the **local** and **global** variables have the same name.

This is called **name overloading**.

We use **`this.`** to distinguish between **local** and **global** variables.



```
Spot.java
public class Spot {
    private float xCoord;
    private float yCoord;
    private float diameter;

    public Spot() {
        xCoord = 100;
        yCoord = 200;
        diameter = 40;
    }

    public Spot(float xCoord, float yCoord, float diameter) {
        this.xCoord = xCoord;
        this.yCoord = yCoord;
        this.diameter = diameter;
    }
}
```

The code shows a Java class named `Spot`. It contains two constructors. The first constructor is a no-argument constructor that initializes `xCoord`, `yCoord`, and `diameter` to 100, 200, and 40 respectively. The second constructor is a parameterized constructor that takes three `float` parameters: `xCoord`, `yCoord`, and `diameter`. Inside this constructor, the local variables `xCoord`, `yCoord`, and `diameter` are assigned values using the `this.` keyword to refer to the global variables defined in the class. A red arrow points from a yellow callout box labeled "Instance fields (global)" to the global `xCoord`, `yCoord`, and `diameter` declarations. Another red arrow points from a green callout box labeled "Parameters (local)" to the local `xCoord`, `yCoord`, and `diameter` declarations in the parameter list of the second constructor.



# Spot Constructor – `this` keyword

Both the **local** and **global** variables have the same name.

This is called **name overloading**.

We use **`this.`** to distinguish between **local** and **global** variables.

```
Spot.java
public class Spot {
    private float xCoord;
    private float yCoord;
    private float diameter;

    public Spot() {
        xCoord = 100;
        yCoord = 200;
        diameter = 40;
    }

    public Spot(float xCoord, float yCoord, float diameter) {
        this.xCoord = xCoord;
        this.yCoord = yCoord;
        this.diameter = diameter;
    }
}
```

Instance fields (global)



# Spot Constructor – `this` keyword

Both the **local** and **global** variables have the same name.

This is called **name overloading**.

We use **`this.`** to distinguish between **local** and **global** variables.

```
Spot.java
public class Spot {
    private float xCoord;
    private float yCoord;
    private float diameter;

    public Spot() {
        xCoord = 100;
        yCoord = 200;
        diameter = 40;
    }

    public Spot(float xCoord, float yCoord, float diameter) {
        this.xCoord = xCoord;
        this.yCoord = yCoord;
        this.diameter = diameter;
    }
}
```

---

# Add a Spot





# Adding a New Spot

```
import java.util.Scanner;  
  
public class Driver {  
  
    Spot spot = new Spot();  
    Scanner input = new Scanner(System.in);
```

Driver class



1

We defined a Scanner object, called **input**, when writing the code to Update a Spot.



# Adding a New Spot

```
import java.util.Scanner;  
  
public class Driver {  
  
    Spot spot = new Spot();  
    Scanner input = new Scanner(System.in);
```

Driver class

2

Now write a new method that will ask the user for the values for the three Spot fields and then use our new constructor to create a new Spot object.

```
void addSpotDetails(){  
    System.out.print("Enter xCoord value: ");  
    float enteredXCoord = input.nextFloat();  
    System.out.print("Enter yCoord value: ");  
    float enteredYCoord = input.nextFloat();  
    System.out.print("Enter diameter value: ");  
    float enteredDiameter = input.nextFloat();  
    spot = new Spot(enteredXCoord, enteredYCoord, enteredDiameter);  
}
```





# Adding a New Spot

```
import java.util.Scanner;  
  
public class Driver {  
  
    Spot spot = new Spot();  
    Scanner input = new Scanner(System.in);
```

Driver class

3

Now call this new method from the  
Driver() constructor

```
Driver(){  
    addSpotDetails(); ←  
    drawSpot();  
    printRadius();  
    printArea();  
    printCircumference();  
    //update spot details and redraw spot  
    updateSpotDetails();  
    drawSpot();  
}
```

# Adding a New Spot

Driver class

```
Run: Driver ×  
/Library/Java/JavaVirtualMachines/  
Enter xCoord value: 34  
Enter yCoord value: 21  
Enter diameter value: 54  
  
xCoord: 34.0  
yCoord: 21.0  
diameter: 54.0  
  
radius: 27.0  
  
area: 2290.1536  
  
circumference: 169.641
```

When you run the app, you should now be asked to enter in any details you wish for each of the fields

4

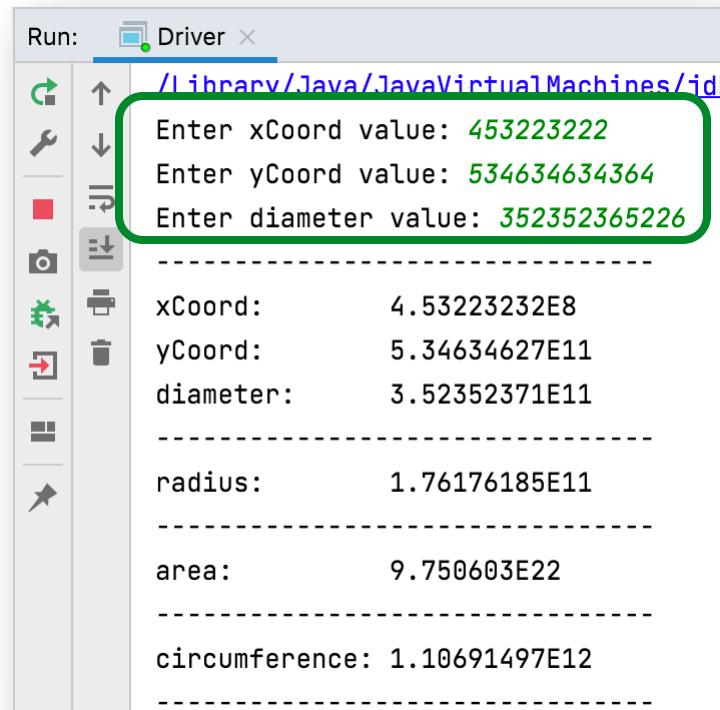
---

# User Input Validation



# No Validation

- ❑ The purpose of this app is to do calculations on a Spot that can be drawn on a monitor.
  
- ❑ Do you, for either the add or the update, try enter absurd values for any of the fields?



The screenshot shows a Java application window titled "Driver". The console output is highlighted with a green rounded rectangle. It displays three user inputs:

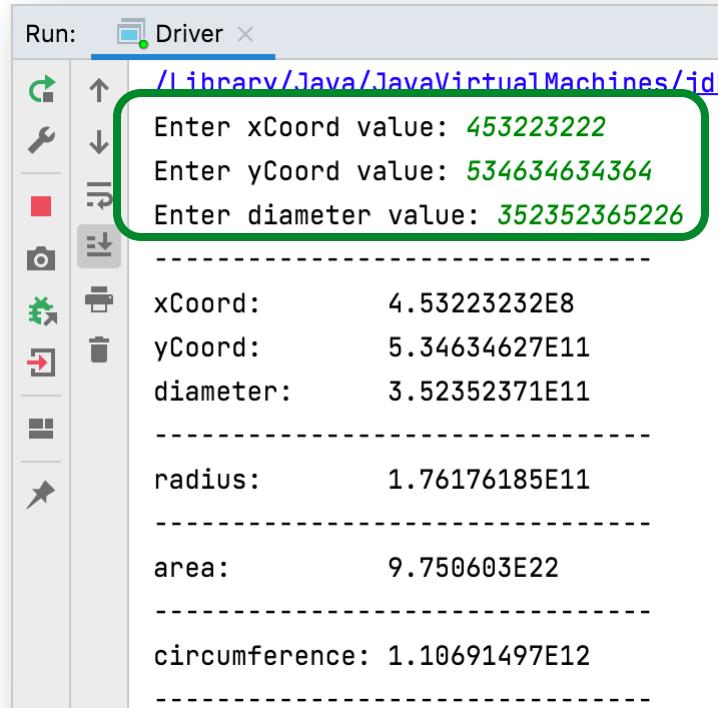
```
Run: Driver
/l ibrary /Java /JavaVirtualMachines /id
Enter xCoord value: 453223222
Enter yCoord value: 534634634364
Enter diameter value: 352352365226
```

Below these inputs, the calculated results are shown:

```
xCoord: 4.53223232E8
yCoord: 5.34634627E11
diameter: 3.52352371E11
-----
radius: 1.76176185E11
-----
area: 9.750603E22
-----
circumference: 1.10691497E12
```

# No Validation

- ❑ Clearly we need to put some restrictions in place for each field i.e.:
  - ❑ min and max permitted values
  - ❑ default value if the min and max are breached.



```
Run: Driver
C:\Users\Java\JavaVirtualMachines\jdk-17.0.1\bin>java Driver
Enter xCoord value: 453223222
Enter yCoord value: 534634634364
Enter diameter value: 352352365226
-----
xCoord: 4.53223232E8
yCoord: 5.34634627E11
diameter: 3.52352371E11
-----
radius: 1.76176185E11
-----
area: 9.750603E22
-----
circumference: 1.10691497E12
-----
```



# Validation Rules

Field	Min Value	Max Value	Default Value
xCoord	Greater than or equal to 0	Less than or equal to 800	400
yCoord	Greater than or equal to 0	Less than or equal to 700	350
diameter	Greater than 0	Less than 600	100



---

# User Input Validation

## Implementing Validation Rules

### Default Values

# Validation Rules – Default Values

Field	Default Value
xCoord	400
yCoord	350
diameter	100

```
c Spot.java ×  
1 public class Spot {  
2  
3     private float xCoord;  
4     private float yCoord;  
5     private float diameter;
```

becomes

```
c Spot.java ×  
1  
2  
3     private float xCoord = 400;  
4     private float yCoord = 350;  
5     private float diameter = 100;
```



---

# User Input Validation

## Implementing Validation Rules

xCoord



# xCoord – mutator changes

Field	Min Value	Max Value
xCoord	Greater than or equal to 0	Less than or equal to 800

```
public void setxCoord(float xCoord) {  
    this.xCoord = xCoord;  
}
```

becomes

```
public void setxCoord(float xCoord) {  
    if ((xCoord >= 0) && (xCoord <= 800)) {  
        this.xCoord = xCoord;  
    }  
}
```



# xCoord – constructor changes

Field	Min Value	Max Value
xCoord	Greater than or equal to 0	Less than or equal to 800

```
public Spot(float xCoord, float yCoord, float diameter) {  
    this.xCoord = xCoord;  
    this.yCoord = yCoord;  
    this.diameter = diameter;  
}
```

becomes

```
public Spot(float xCoord, float yCoord, float diameter) {  
    setxCoord(xCoord);  
    this.yCoord = yCoord;  
    this.diameter = diameter;  
}
```



---

# User Input Validation

## Implementing Validation Rules

yCoord



# yCoord – mutator changes

Field	Min Value	Max Value
yCoord	Greater than or equal to 0	Less than or equal to 700

```
public void setyCoord(float yCoord) {  
    this.yCoord = yCoord;  
}
```



```
public void setyCoord(float yCoord) {  
    if ((yCoord >= 0) && (yCoord <= 700)) {  
        this.yCoord = yCoord;  
    }  
}
```



# yCoord – constructor changes

Field	Min Value	Max Value
yCoord	Greater than or equal to 0	Less than or equal to 700

```
public Spot(float xCoord, float yCoord, float diameter) {  
    setxCoord(xCoord);  
    this.yCoord = yCoord;  
    this.diameter = diameter;  
}
```

becomes

```
public Spot(float xCoord, float yCoord, float diameter) {  
    setxCoord(xCoord);  
    setyCoord(yCoord);  
    this.diameter = diameter;  
}
```



---

# User Input Validation

## Implementing Validation Rules

diameter



# diameter – mutator changes

Field	Min Value	Max Value
diameter	Greater than 0	Less than 600

```
public void setDiameter(float diameter) {  
    this.diameter = diameter;  
}
```



```
public void setDiameter(float diameter) {  
    if ((diameter > 0) && (diameter < 600)) {  
        this.diameter = diameter;  
    }  
}
```



# diameter – constructor changes

Field	Min Value	Max Value
diameter	Greater than 0	Less than 600

```
public Spot(float xCoord, float yCoord, float diameter) {  
    setxCoord(xCoord);  
    setyCoord(yCoord);  
    this.diameter = diameter;  
}
```

becomes

```
public Spot(float xCoord, float yCoord, float diameter) {  
    setxCoord(xCoord);  
    setyCoord(yCoord);  
    setDiameter(diameter);  
}
```



---

# User Input Validation

Boundary Testing the Validation Rules

for Add and Update



# What is Boundary Testing?

- ❑ Boundary Testing is when you input test data that is:
    - Just inside
    - Just outside
- the boundary values in your Boolean expressions.

Inputting the following values for xCoord would test the ‘boundaries’ of this if statement:

[-1, 0, 1, 799, 800, 801]

```
public void setxCoord(float xCoord) {  
    if ((xCoord >= 0) && (xCoord <= 800)) {  
        this.xCoord = xCoord;  
    }  
}
```

# Example Boundary Test

## Just-Outside Upper Boundary Test:

- default values are correctly used when adding and updating

```
Run: Driver < /Library/Java/JavaVirtualMachines/jdk  
Enter xCoord value: 801  
Enter yCoord value: 701  
Enter diameter value: 600  
  
xCoord: 400.0  
yCoord: 350.0  
diameter: 100.0  
  
radius: 50.0  
  
area: 7853.75  
  
circumference: 314.15  
  
Enter new xCoord value: 801  
Enter new yCoord value: 701  
Enter new diameter value: 600  
  
xCoord: 400.0  
yCoord: 350.0  
diameter: 100.0  
  
Process finished with exit code 0
```

# Example Boundary Test

## Just-Inside Upper Boundary Test:

- all values are accepted for both add and update

```
Run: Driver >
/Library/Java/JavaVirtualMachines/jdk
Enter xCoord value: 800
Enter yCoord value: 700
Enter diameter value: 599
xCoord: 800.0
yCoord: 700.0
diameter: 599.0
radius: 299.5
area: 281793.34
circumference: 1881.7585
Enter new xCoord value: 800
Enter new yCoord value: 700
Enter new diameter value: 599
xCoord: 800.0
yCoord: 700.0
diameter: 599.0
Process finished with exit code 0
```

# Example Boundary Test

## Just-Inside Lower Boundary Test:

- all values are accepted for both add and update

```
Run: Driver < /Library/Java/JavaVirtualMachines/jdk  
Enter xCoord value: 0  
Enter yCoord value: 0  
Enter diameter value: 1  
  
xCoord: 0.0  
yCoord: 0.0  
diameter: 1.0  
  
radius: 0.5  
  
area: 0.785375  
  
circumference: 3.1415  
  
Enter new xCoord value: 0  
Enter new yCoord value: 0  
Enter new diameter value: 1  
  
xCoord: 0.0  
yCoord: 0.0  
diameter: 1.0  
  
Process finished with exit code 0
```

# Questions?

---



# Thanks.

