Programming Fundamentals 1

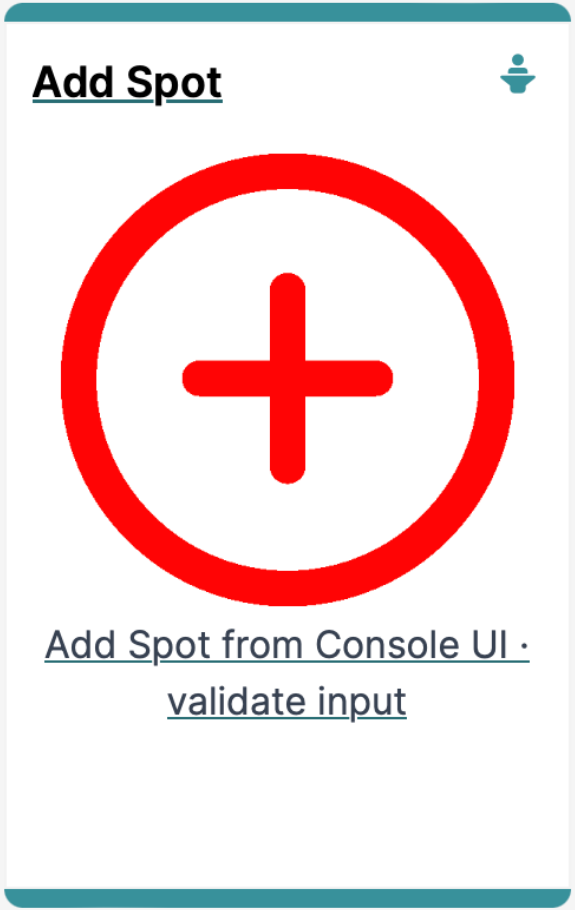
Produced by Mr. Dave Drohan (david.drohan@setu.ie)
Dr. Siobhán Drohan
Ms. Mairead Meagher

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

setu.ie

IntelliJ and Spot

Add Spot (and Validate Input)

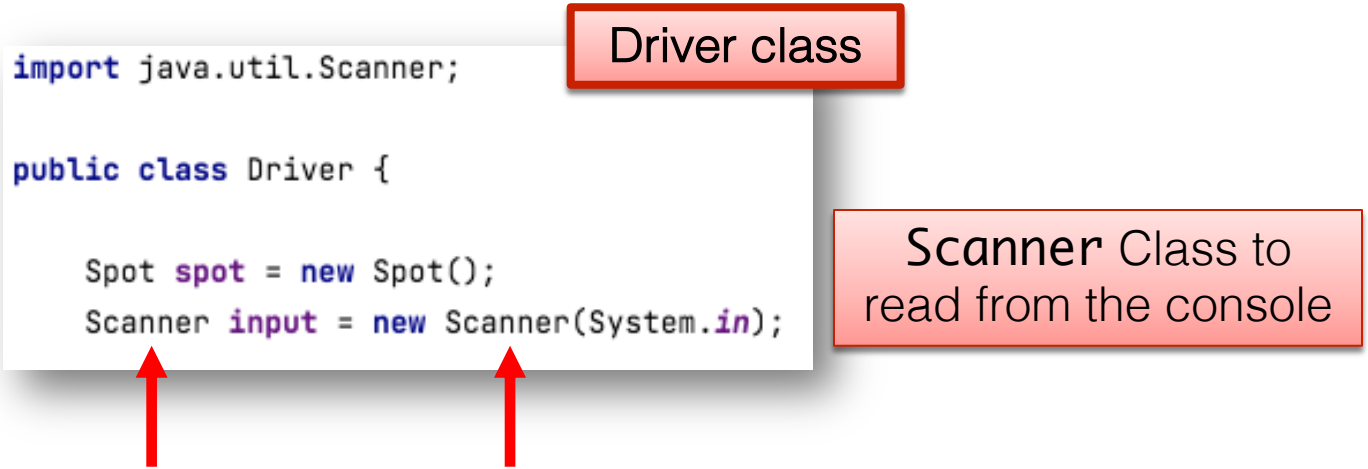


Agenda

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

RECAP

RECAP: Scanner class



RECAP: `Driver` class

Driver class

```
import java.util.Scanner;

public class Driver {

    Spot spot = new Spot();
    Scanner input = new Scanner(System.in);
```

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

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     }
```

Spot Constructor – adding a second one

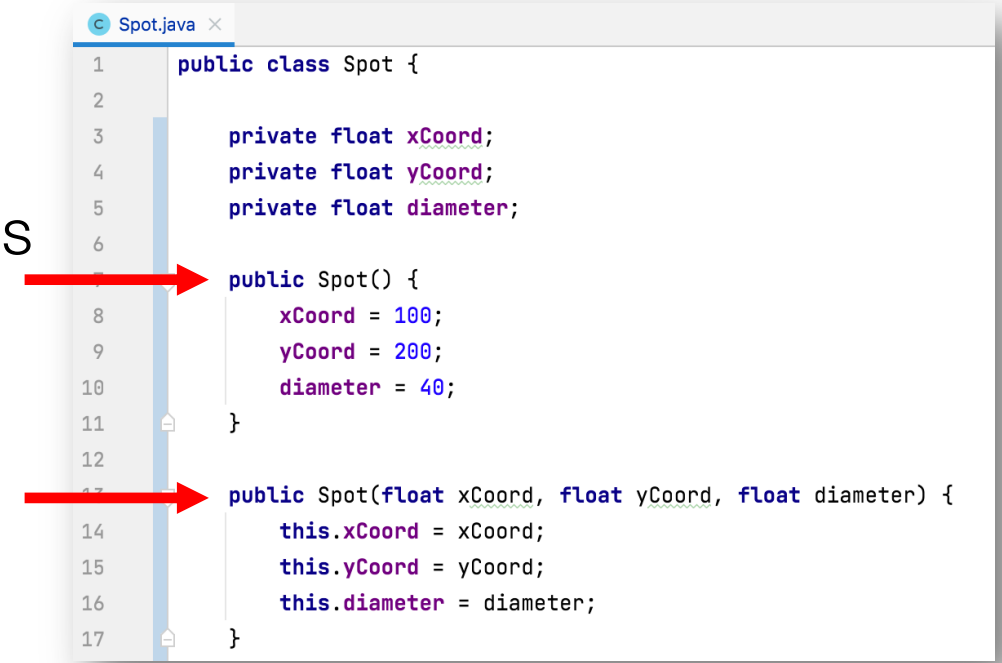
- If we want to allow the user to add a new Spot with their own data (read from the console), then we need to provide a second constructor that will allow this

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 Constructor – overloading

- Why would we need two different constructors?

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

- 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

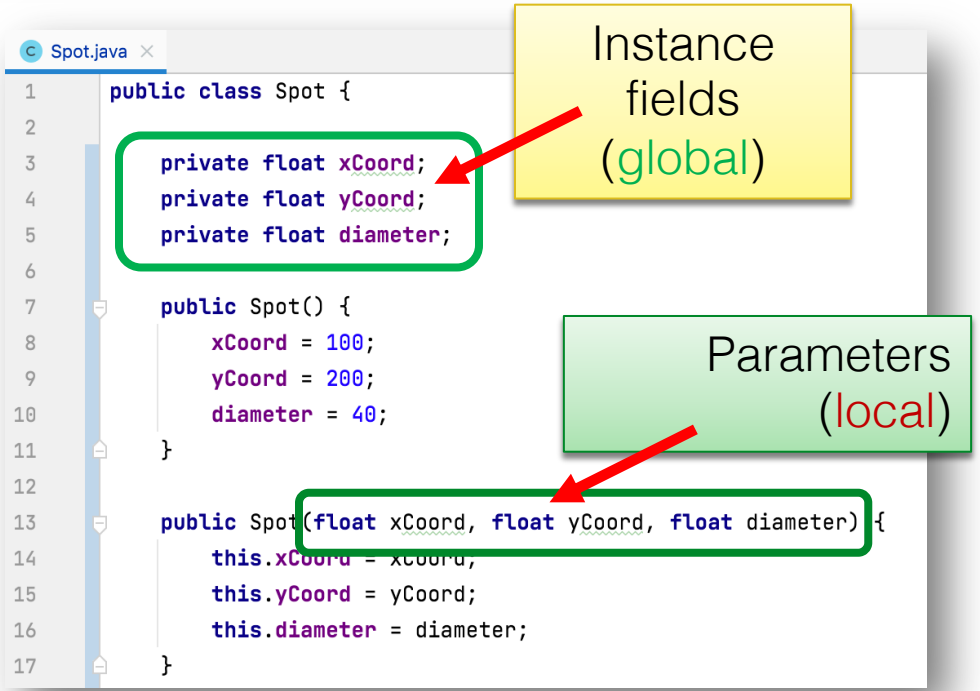
```
1   public class Spot {
2
3       private float xCoord;
4       private float yCoord;
5       private float diameter;
6
7       public Spot() {
8           xCoord = 100;
9           yCoord = 200;
10          diameter = 40;
11      }
12
13      public Spot(float xCoord, float yCoord, float diameter)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

Spot Constructor – `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 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

```
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     }
```

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

```
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      }
```

Parameters (local)

Add a Spot

Adding a New Spot

Driver class

```
import java.util.Scanner;

public class Driver {

    Spot spot = new Spot();
    Scanner input = new Scanner(System.in);
```

1

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

Adding a New Spot

Driver class

```
import java.util.Scanner;

public class Driver {

    Spot spot = new Spot();
    Scanner input = new Scanner(System.in);
```

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

Driver class

```
import java.util.Scanner;

public class Driver {

    Spot spot = new Spot();
    Scanner input = new Scanner(System.in);
```

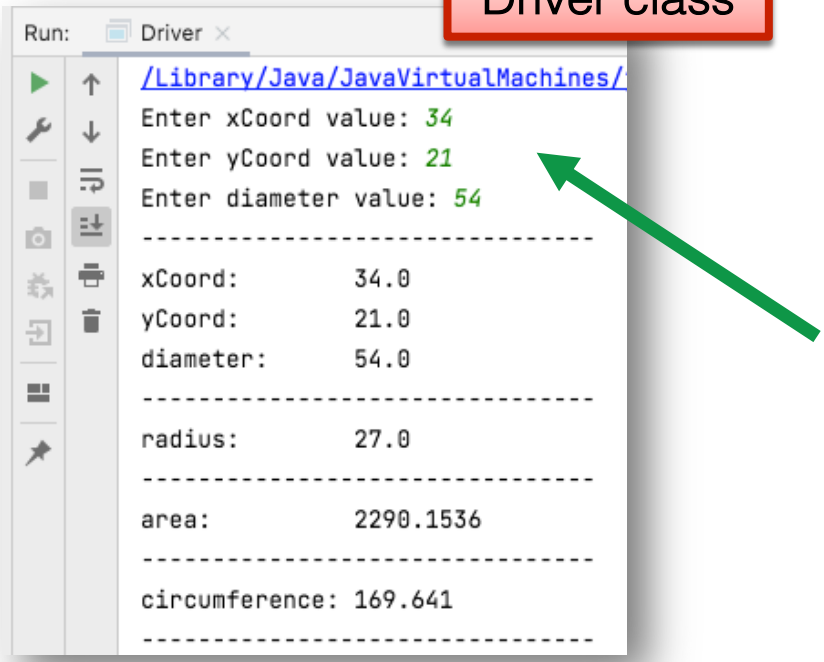
3

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



4

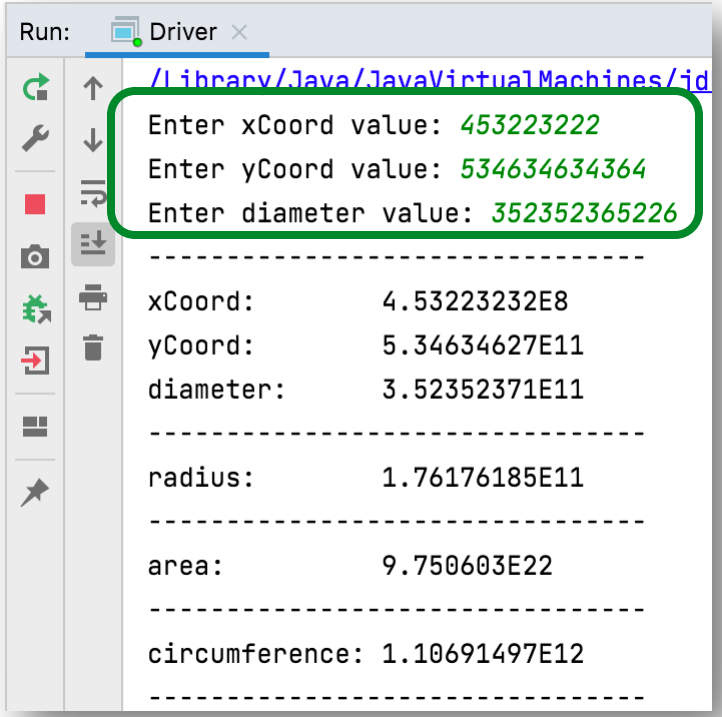
When you run the app, you should now be asked to enter in any details you wish for each of the fields

User Input Validation

No Validation

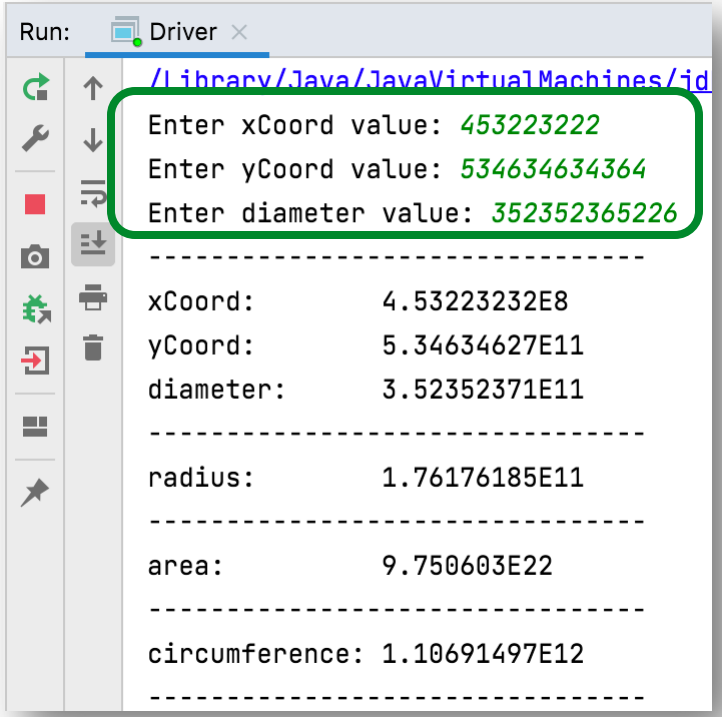
- The purpose of this app is to do calculations on a Spot that can be drawn on a monitor.

- Do you, for either the add or the update, try enter absurd values for any of the fields?



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.



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

Spot.java

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

becomes

Spot.java

```
1  public class Spot {
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;
}
```

becomes

```
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;
}
```

becomes

```
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:

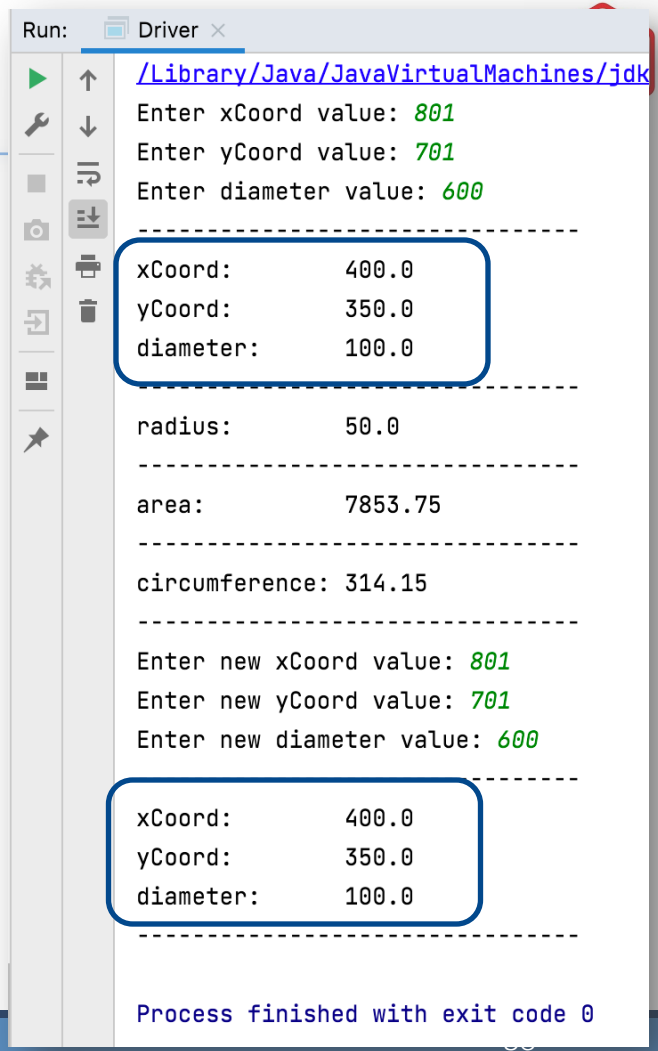
[-1, 0, 1, 799, 800, 801]

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

Example Boundary Test

Just-Outside Upper Boundary Test:

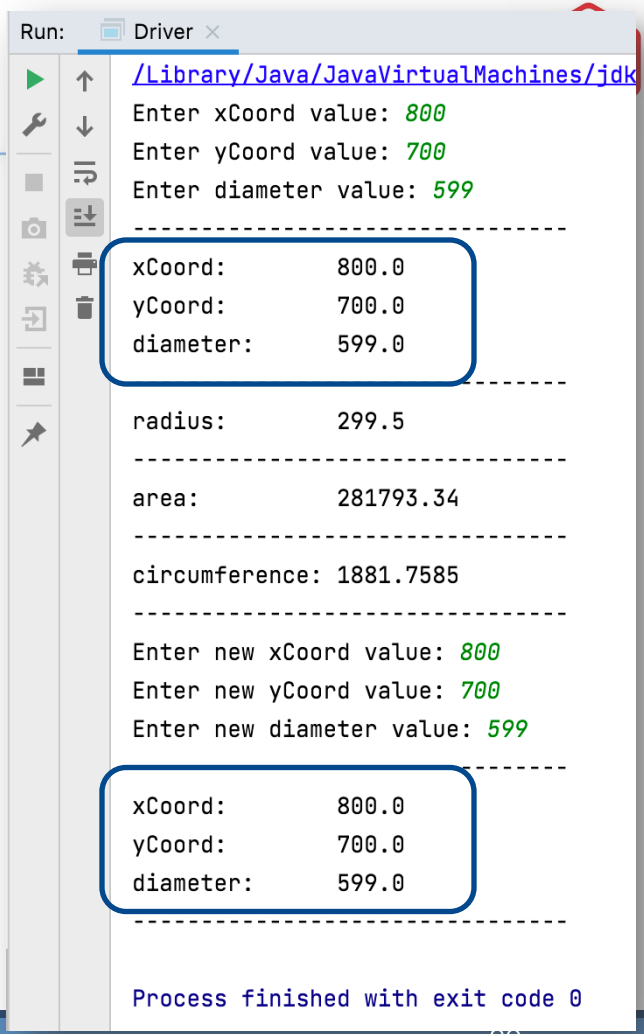
- default values are correctly used when adding and updating



Example Boundary Test

Just-Inside Upper Boundary Test:

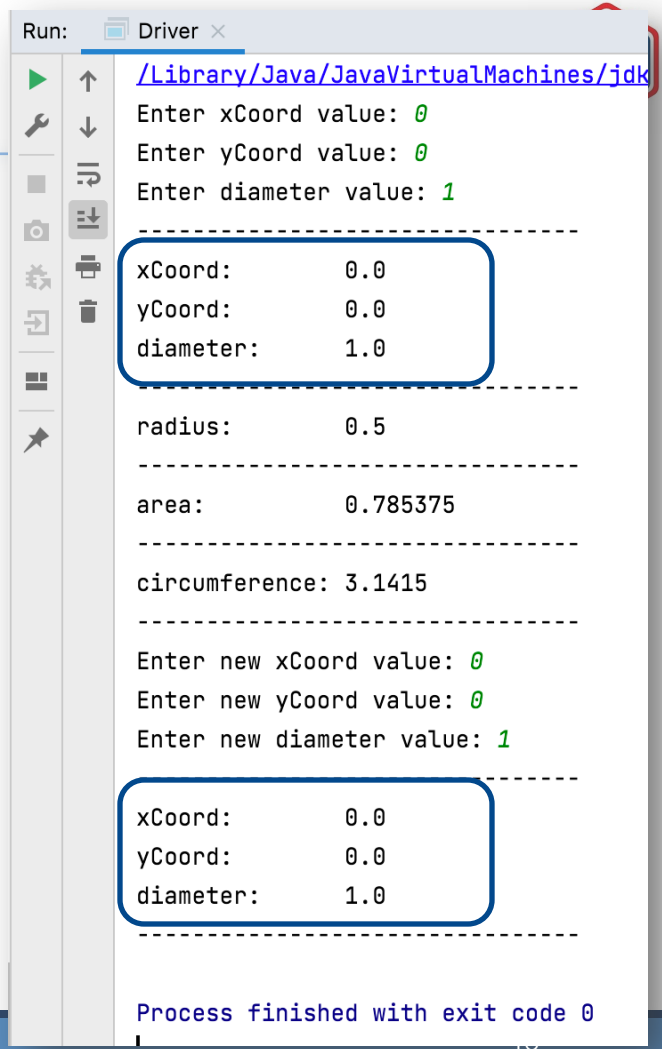
- all values are accepted for both add and update



Example Boundary Test

Just-Inside Lower Boundary Test:

- all values are accepted for both add and update



Questions?


Thanks.