



Northeastern  
University

# Lecture 8: Object Oriented Programming - 3

Prof. Chen-Hsiang (Jones) Yu, Ph.D.  
College of Engineering

Materials are edited by Prof. Jones Yu from

Liang, Y. Daniel. Introduction to Java Programming and Data Structures, Comprehensive  
Version, 12th edition, Pearson, 2019.

# Outline

---

- Objects and Classes
- Thinking in Objects

# Objects and Classes

## Passing Objects to Methods

# Passing Objects to Methods

---

- Passing by **value** for **primitive type value**
  - » The value is passed to the parameter.
- Passing by **value** for **reference type value**
  - » The value is the **reference to the object**.

```

public class CircleWithPrivateDataFields {
    /** The radius of the circle */
    private double radius = 1;

    /** The number of the objects created */
    private static int numberOfObjects = 0;

    /** Construct a circle with radius 1 */
    public CircleWithPrivateDataFields() {
        numberOfObjects++;
    }

    /** Construct a circle with a specified radius */
    public CircleWithPrivateDataFields(double newRadius)
    {
        radius = newRadius;
        numberOfObjects++;
    }

    /** Return radius */
    public double getRadius() {
        return radius;
    }

    /** Set a new radius */
    public void setRadius(double newRadius) {
        radius = (newRadius >= 0) ? newRadius : 0;
    }

    /** Return numberOfObjects */
    public static int getNumberOfObjects() {
        return numberOfObjects;
    }

    /** Return the area of this circle */
    public double getArea() {
        return radius * radius * Math.PI;
    }
}

```

```

public class TestPassObject {
    public static void main(String[] args) {
        // Create a Circle object with radius 1
        CircleWithPrivateDataFields myCircle =
            new CircleWithPrivateDataFields(1);

        // Print areas for radius 1, 2, 3, 4, and 5.
        int n = 5;
        printAreas(myCircle, n);

        // See myCircle.radius and times
        System.out.println("\n" + "Radius is " +
            myCircle.getRadius());
        System.out.println("n is " + n);
    }

    /** Print a table of areas for radius */
    public static void
    printAreas(CircleWithPrivateDataFields c, int times) {
        System.out.println("Radius \t\tArea");
        while (times >= 1) {
            System.out.println(c.getRadius() + "\t\t" + c.getArea());
            c.setRadius(c.getRadius() + 1);
            times--;
        }
    }
}

```

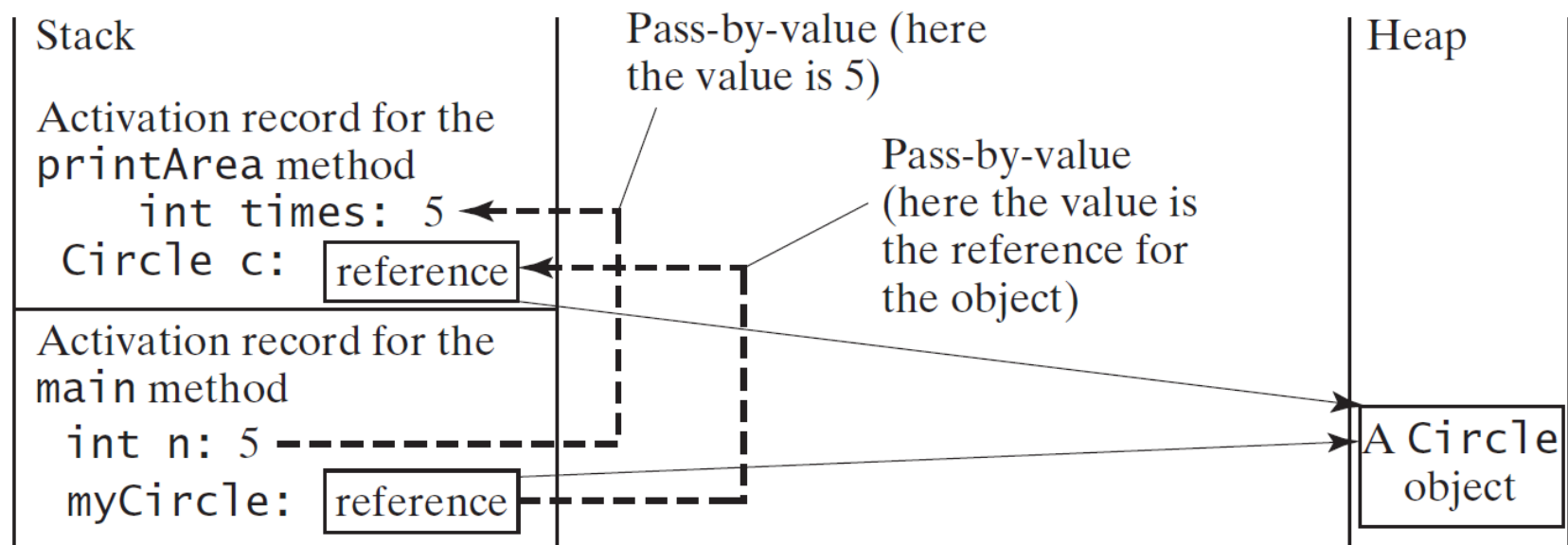
Radius	Area
1.0	3.141592653589793
2.0	12.566370614359172
3.0	28.274333882308138
4.0	50.26548245743669
5.0	78.53981633974483

```

Radius is 6.0
n is 5

```

# Passing Objects to Methods



Array of Objects



# Array of Objects

---

- An array of objects is actually an *array of reference variables*.

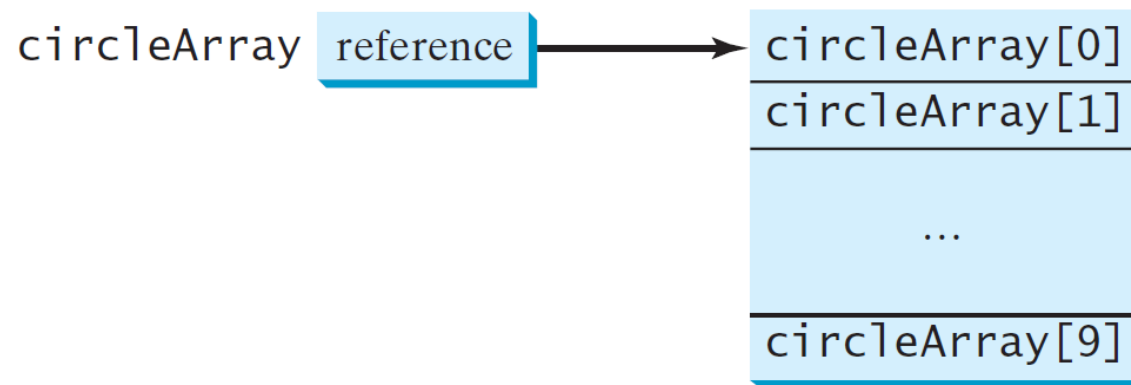
```
Circle[] circleArray = new Circle[10];
```

- So invoking `circleArray[1].getArea()` involves two levels of referencing as shown in the next figure.
  - » `circleArray` references to the entire array.
  - » `circleArray[1]` references to a `Circle` reference variable that points/refers to a `Circle` object.

# Array of Objects

---

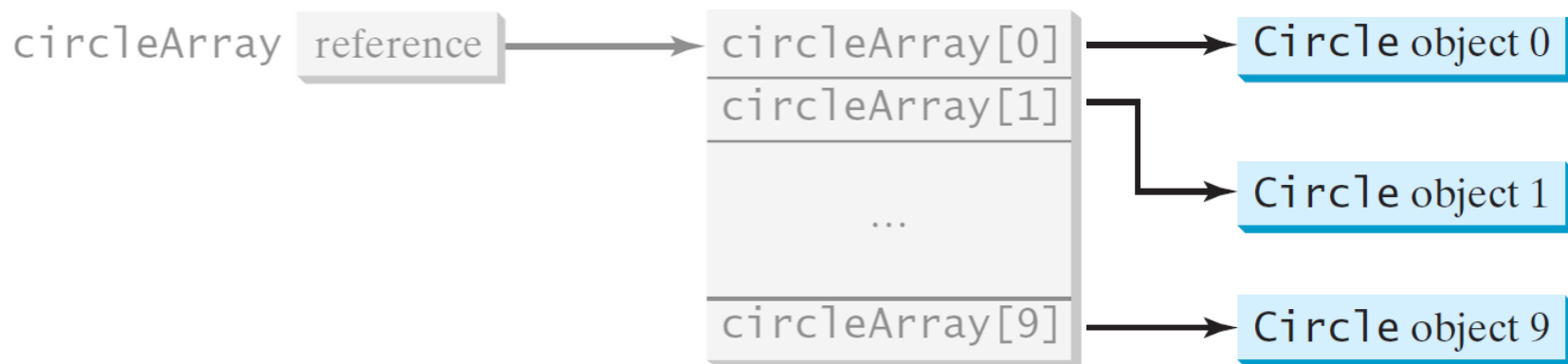
```
Circle[] circleArray = new Circle[10];
```



## Array of Objects (cont.)

---

```
Circle[] circleArray = new Circle[10];  
  
for (int i = 0; i < 10; i++) {  
    circleArray[i] = new Circle();  
}
```



# Exercise

---

- What is wrong in the following code? Please type it into your Eclipse Java project.
- Please fix it and print out the value for current time.

```
import java.util.Date;

public class ClassExamples {
    public static void main(String[] args){
        Date[] dates = new Date[10];

        System.out.println(dates[0]);
        System.out.println(dates[0].toString());
    }
}
```

# Answer

---

```
import java.util.Date;

public class Objects_and_Classes_Lecture3 {
    public static void main(String[] args){
        Date[] dates = new Date[10];

        for(int i = 0; i < 10; i++){
            dates[i] = new Date();
        }

        System.out.println(dates[0]);
        System.out.println(dates[0].toString());
    }
}
```

# The **this** Reference

# This **this** Keyword

---

- The **this** keyword is the name of a reference that refers to an object itself.
- Common uses:
  - » It can also be used inside a constructor to **invoke another constructor** of the same class.
  - » **this** keyword is a reference to a **class's *hidden* (private) *data fields***.

# Reference the Hidden Data Fields

---

```
public class F {  
    private int i = 5;  
    private static double k = 0;  
  
    void setI(int i) {  
        this.i = i;  
    }  
  
    static void setK(double k) {  
        F.k = k;  
    }  
}
```

Suppose that f1 and f2 are two objects of F.

```
F f1 = new F(); F f2 = new F();
```

Invoking f1.setI(10) is to execute

```
this.i = 10, where this refers f1
```

Invoking f2.setI(45) is to execute


```
this.i = 45, where this refers f2
```




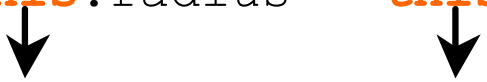
# Calling Overloaded Constructor

---

```
public class Circle {  
    private double radius;  
  
    public Circle(double radius) {  
        this.radius = radius;  
    }  
  
    public Circle() {  
        this(1.0);  
    }  
  
    public double getArea() {  
        return this.radius * this.radius * Math.PI;  
    }  
}
```

 this must be explicitly used to reference the data field radius of the object being constructed

 this is used to invoke another constructor

 Every instance variable belongs to an instance represented by this, which is normally omitted

## Exercise

---

- Write a class named `Account` which has two (`private`) data fields: the `balance` of the account and the `name` of the account
- Include `a constructor` that allows you to set the name and the initial balance of the account
- Also include a `toString()` method that puts both the name and balance in the String
- Write a `main()` method to test the class

# Answer

---

Account.java:

```
public class Account {  
    private String name;  
    private double balance;  
  
    public Account(String accountName, double initialBalance) {  
        this.name = accountName;  
        this.balance = initialBalance;  
    }  
  
    public String toString() {  
        String output = name;  
        output += String.format(": $%.2f", balance);  
        return output;  
    }  
}
```

MyTest.java:

```
public class MyTest {  
    public static void main(String[] args) {  
        Account checking = new Account("Checking", 0.93);  
        System.out.println(checking);  
    }  
}
```

## Exercise (Continued) - Offline

---

- Modify your `Account` class to include a default constructor that sets the `balance` to \$0.00 and the `name` to "Account".
- Also add a method named `adjust()` that allows you to adjust the balance by a positive or negative amount.
- Test the new methods in `main()`.

# Answer

---

Account.java:

```
public class Account {
    private String name;
    private double balance;

    public Account() {
        this.name = "Account";
        this.balance = 0;
    }

    public Account(String accountName, double initialBalance) {
        this.name = accountName;
        this.balance = initialBalance;
    }

    public void adjust(double amount) {
        balance = balance + amount;
    }

    public String toString() {
        String output = name;
        output += String.format(": $%.2f", balance);
        return output;
    }
}
```

MyTest.java:

```
public class MyTest {
    public static void main(String[] args) {
        Account checking = new Account("Checking", 0.93);
        System.out.println(checking);
        Account account = new Account();
        account.adjust(1000);
        account.adjust(-250);
        System.out.println(account);
    }
}
```

# Thinking in Objects

# Class Abstraction and Encapsulation

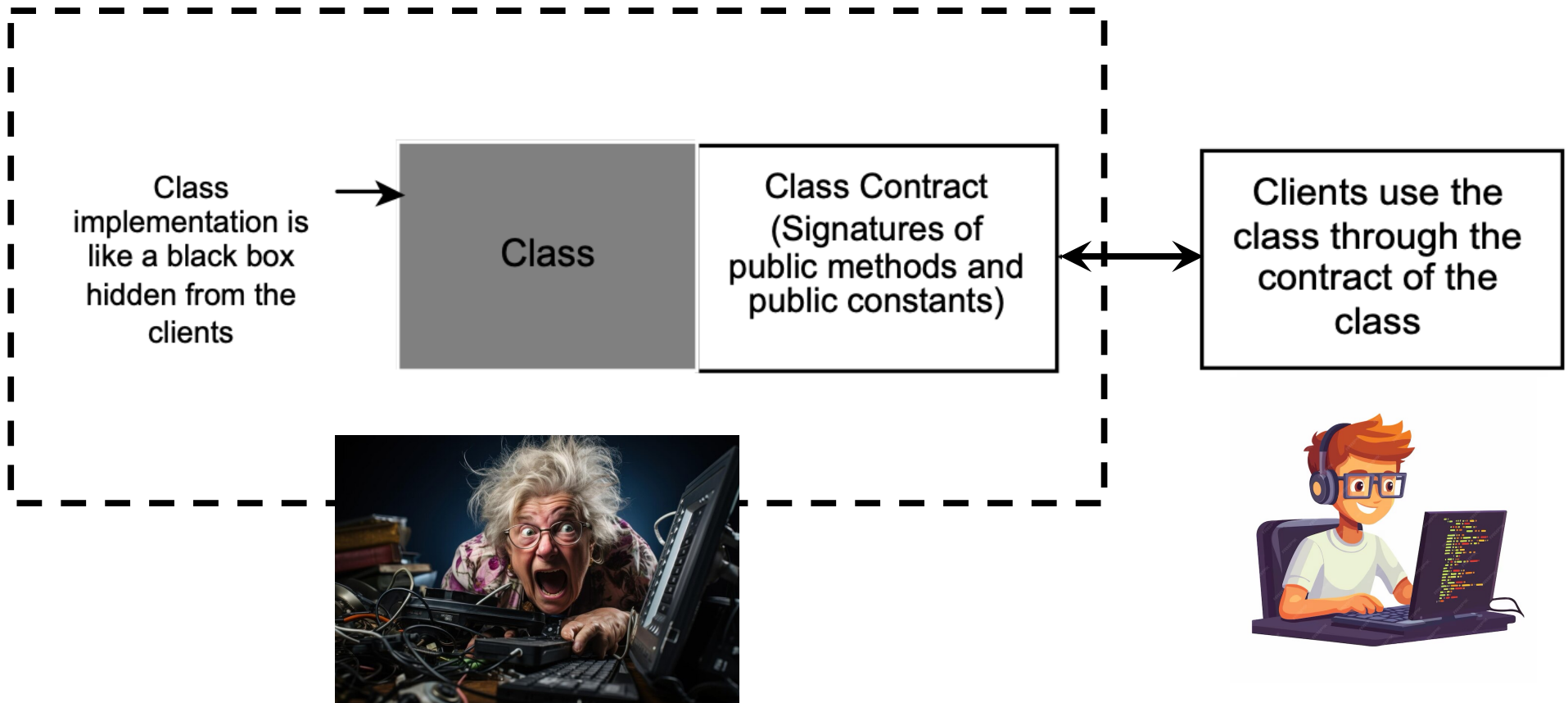
# Class Abstraction and Encapsulation

---

- Class abstraction means to separate **class implementation** from **the use of the class**.
- **The creator** of the class provides a description of the class and let **the user** know how the class can be used.
- **The user** of the class does not need to know how the class is implemented. **The detail of implementation is encapsulated and hidden** from the user.



# Class Abstraction and Encapsulation



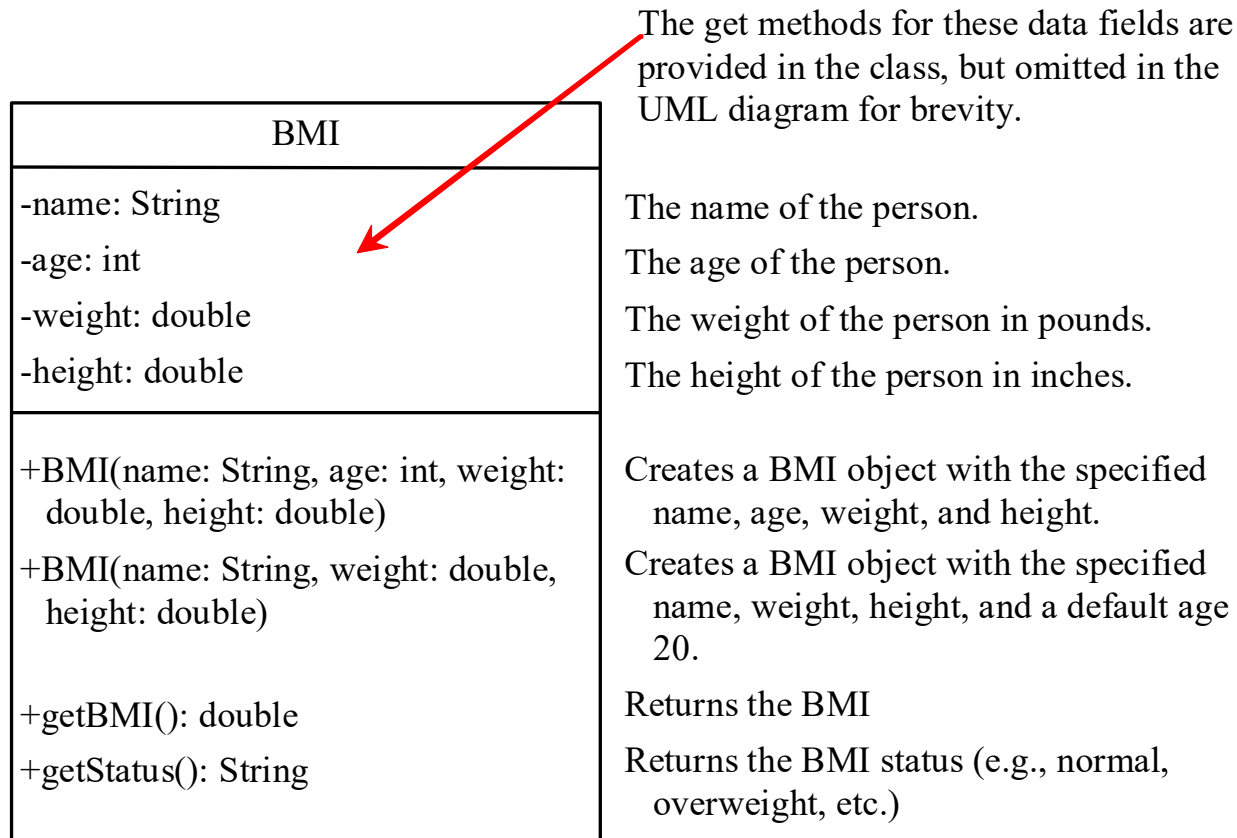
Images:

[https://www.freepik.com/premium-ai-image/busy-senior-programmer-getting-mad-messy-table-generative-ai\\_61370132.htm](https://www.freepik.com/premium-ai-image/busy-senior-programmer-getting-mad-messy-table-generative-ai_61370132.htm)

[https://www.freepik.com/premium-vector/professional-software-developer-coding-with-multiple-computer-languages\\_343650584.htm#fromView=search&page=1&position=29&uuid=e4b45a3e-9019-47c1-bc53-4fc434c4c0a0](https://www.freepik.com/premium-vector/professional-software-developer-coding-with-multiple-computer-languages_343650584.htm#fromView=search&page=1&position=29&uuid=e4b45a3e-9019-47c1-bc53-4fc434c4c0a0)

# Case Study: The BMI Class

---



## BMI.java

```
public class BMI {
    private String name;
    private int age;
    private double weight; // in pounds
    private double height; // in inches

    public static final double KILOGRAMS_PER_POUND = 0.45359237;
    public static final double METERS_PER_INCH = 0.0254;

    public BMI(String name, int age, double weight, double height) {
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.height = height;
    }

    public BMI(String name, double weight, double height) {
        this(name, 20, weight, height);
    }

    public double getBMI() {
        double bmi = weight * KILOGRAMS_PER_POUND / ((height * METERS_PER_INCH) * (height * METERS_PER_INCH));
        return Math.round(bmi * 100) / 100.0;
    }

    public String getStatus() {
        double bmi = getBMI();
        if (bmi < 18.5)
            return "Underweight";
        else if (bmi < 25)
            return "Normal";
        else if (bmi < 30)
            return "Overweight";
        else
            return "Obese";
    }

    public String getName() { return name; }
    public int getAge() { return age; }
    public double getWeight() { return weight; }
    public double getHeight() { return height; }
}
```

## UseBMIClass.java

```
public class UseBMIClass {  
    public static void main(String[] args) {  
        BMI bmi1 = new BMI("John Doe", 18, 145, 70);  
        System.out.println("The BMI for " + bmi1.getName() + " is "  
            + bmi1.getBMI() + " " + bmi1.getStatus());  
  
        BMI bmi2 = new BMI("Peter King", 215, 70);  
        System.out.println("The BMI for " + bmi2.getName() + " is "  
            + bmi2.getBMI() + " " + bmi2.getStatus());  
    }  
}
```

```
The BMI for John Doe is 20.81 Normal  
The BMI for Peter King is 30.85 Obese
```