King Saud University
College of Computer & Information Science
CSC111 – Tutorial11
Object – III –
All Sections

Objectives:

- To develop methods with object arguments and differentiate between primitive-type arguments and object-type arguments.
- To determine the scope of variables in the context of a class.
- To use the keyword this to refer to the calling object itself.

(***Exercise with * can be left to the student as self review questions***)

Exercise 1

1. * What is the output of the following program (program is separated into two files C.java and TestC.java)?

```
public class C {
    private int p;

    public C(int newP) {
        p = newP;
    }

    public int getP() {
        return p;
    }

    public void setP(int newP) {
        p = newP;
    }
}
```

Figure 1: C.java

```
public class TestC {
    public static void main(String[] args) {
        C f = new C(1);
        int a = 10;
        changePrimitiveVar(a);
        System.out.println("a = " + a);
        changeReference(f);
        System.out.println("f.p = " + f.getP());
        f = createNewObject();//
        System.out.println("f.p = " + f.getP());
        f = createNewObject2();
        System.out.println("f.p = " + f.getP());
        modifyReference(f);
        System.out.println("f.p = " + f.getP());
    }

    public static void changePrimitiveVar(int v) {
        v = 20;
    }

    public static void changeReference(C a) {
        C b = new C(2);
        a = b;
    }

    // a factory method
    public static C createNewObject() {
        return new C(3);
    }

    // another factory method
    public static C createNewObject2() {
        C x = new C(4);
        x.setP(5);
        return x;
    }

    public static void modifyReference(C c) {
        c.setP(6);
        C d = c;
        d.setP(7);
    }
}
```

Figure 2: TestC.java

2. * What is the output of the following program?

```
public class Test {
  public static void main(String[] args) {
    Count myCount = new Count();
    int times = 0;

    for (int i = 0; i < 100; i++)
      increment(myCount, times);

    System.out.println("count is " + myCount.count);
    System.out.println("times is " + times);
  }

  public static void increment(Count c, int times) {
    c.count++;
    times++;
  }
}
```

```
public class Count {
  public int count;

  public Count(int c) {
    count = c;
  }

  public Count() {
    count = 1;
  }
}
```

3. What is the output of the following program:

```
public class TestCircle {
    public static void main(String[] args) {
        Circle circle1 = new Circle(1);
        Circle circle2 = new Circle(2);
        swap1(circle1, circle2);
        System.out.println("After swap1: circle1 = " + circle1.radius
                + " circle2 = " + circle2.radius);
        swap2(circle1, circle2);
        System.out.println("After swap2: circle1 = " + circle1.radius
                + " circle2 = " + circle2.radius);
    }

    public static void swap1(Circle x, Circle y) {
        Circle temp = x;
        x = y;
        y = temp;
    }

    public static void swap2(Circle x, Circle y) {
        double temp = x.radius;
        x.radius = y.radius;
        y.radius = temp;
    }
}

class Circle {
    double radius;

    Circle(double newRadius) {
        radius = newRadius;
    }
}
```

4. What is the output of the following program:

```
public class Test {
  private static int i = 0;
  private static int j = 0;

  public static void main(String[] args) {
    int i = 2;
    int k = 3;

    {
      int j = 3;
      System.out.println("i + j is " + i + j);
    }

    k = i + j;
    System.out.println("k is " + k);
    System.out.println("j is " + j);
  }
}
```

5. What is the output of the following program?

```
class Foo {
    private int a;
    private int b;
    public Foo(){
        this(1, 2);
    }
    public Foo(int a){
        this(a, 12);
    }
    public Foo(int a, int b){
        this.a = a;
        this.b = b;
    }

    public void setA(int a){
        this.a = a;
    }

    public void setB(int b){
        b = b;
    }

    public int getA(){
        return a;
    }

    public int getB(){
        return b;
    }
}

public class TestThisReference {
    public static void main(String[] args) {
        Foo f1 = new Foo(10, 20);
        Foo f2 = new Foo();
        Foo f3 = new Foo(11);
        System.out.println("f1.a = " + f1.getA() +
                ", f1.b = " + f1.getB() + ", f2.a = " +
                f2.getA() + ", f2.b = " + f2.getB() +
                ", f3.a = " + f3.getA() + ", f3.b = " +
                f3.getB());

        f1.setA(5);
        f1.setB(6);
        f2.setA(7);
        f2.setB(8);
        System.out.println("f1.a = " + f1.getA() +
                ", f1.b = " + f1.getB() + ", f2.a = " +
                f2.getA() + ", f2.b = " + f2.getB());
    }
}
```

6. * We know that the width and length of a rectangle cannot be a negative value (i.e., there is no rectangle of length -2). When designing a class Rectangle, we need to enforce this fact and make sure that all objects of type Rectangle do not get into an invalid state in which the length and/or width is negative. To achieve this we use information hiding (private attributes) and setter methods that validate input before mutating the state of the object. Given this, what is wrong with the following Rectangle class and how to correct the problem?

```
class Rectangle{
    private int width;
    private int length;
    public Rectangle(){
        this(1,1);
    }
    public Rectangle(int w, int l){
        length = l; width = w;
    }
    public void setLength(int l){
        if (l >= 0){
            length = l;
        }
    }
    public void setWidth(int w){
        if (w >= 0){
            width = w;
        }
    }
    public int getLength(){
        return length;
    }
    public int getWidth(){
        return width;
    }
}
```

Solution

1)

OUTPUT

```
a = 10
f.p = 1
f.p = 3
f.p = 5
f.p = 7
```

2)

OUTPUT

```
count is 101
times is 0
```

3)

OUTPUT

```
After swap1: circle1 = 1.0 circle2 = 2.0
After swap2: circle1 = 2.0 circle2 = 1.0
```

4)

OUTPUT

```
f1.a = 10, f1.b = 20, f2.a = 1, f2.b = 2, f3.a = 11, f3.b = 12
f1.a = 5, f1.b = 20, f2.a = 7, f2.b = 2
```

5)

The setter methods validate the length before mutating the state but the constructor does not do this. This allows user of the class to create an object with an invalid state from the beginning.
Best solution is for the constructor to call setter methods, which will result in a single location for validating each input.
Here is the class after doing this change:

```java
public class Rectangle{
    private int width;
    private int length;
    public Rectangle(){
        this(1,1);
    }
    public Rectangle(int w, int l){
        setLength(l);
        setWidth(w);
    }
    public void setLength(int l){
        if (l >= 0){
            length = l;
        }
    }
    public void setWidth(int w){
        if (w >= 0){
            width = w;
        }
    }
    public int getLength(){
        return length;
    }
    public int getWidth(){
        return width;
    }
}
```

Exercise 2

We would like to design a class called `Rational` for performing arithmetic with fractions.

- It has two private instance variables—the `numerator` and the `denominator`. (*Hint*: Remember that denominator cannot be 0).
- A constructor that enables an object of this class to be initialized to values provided by the user when it is declared. This constructor should call the setter method to validate the data.
- A no-argument constructor with default values 1 and 1 in case no initializers are provided. This constructor should call the previous constructor.
- A number of public methods that perform each of the following operations (in the following discussion, *current rational number* is the object of type Rational that is used to call the method):
    - `setNumAndDenom(int numerator, int denominator)` which sets the values of the numerator and denominator *after validating them* if necessary.
    - `add(Rational right)` returns a new `Rational` object that contains the result of adding current rational number and rational number right.
    - `subtract(Rational right)` returns a new `Rational` object that contains the result of subtracting rational number right from current object.
    - `multiply(Rational right)` returns a new `Rational` object that contains the result of multiplying current rational number by rational number right.

- `divide(Rational right)` returns a new `Rational` object that contains the result of dividing current rational number by rational number right.
- `print()` which prints current rational number in the form a/b, where a is the numerator and b is the denominator.
- `printFloat()` which prints current rational number in floating-point format.
- `toString()` which returns the string "a/b" where a is the numerator and b is the denominator.

Draw the UML for the class then implement the class. Write a program `TestRational` to test your class. Your program should do the following:

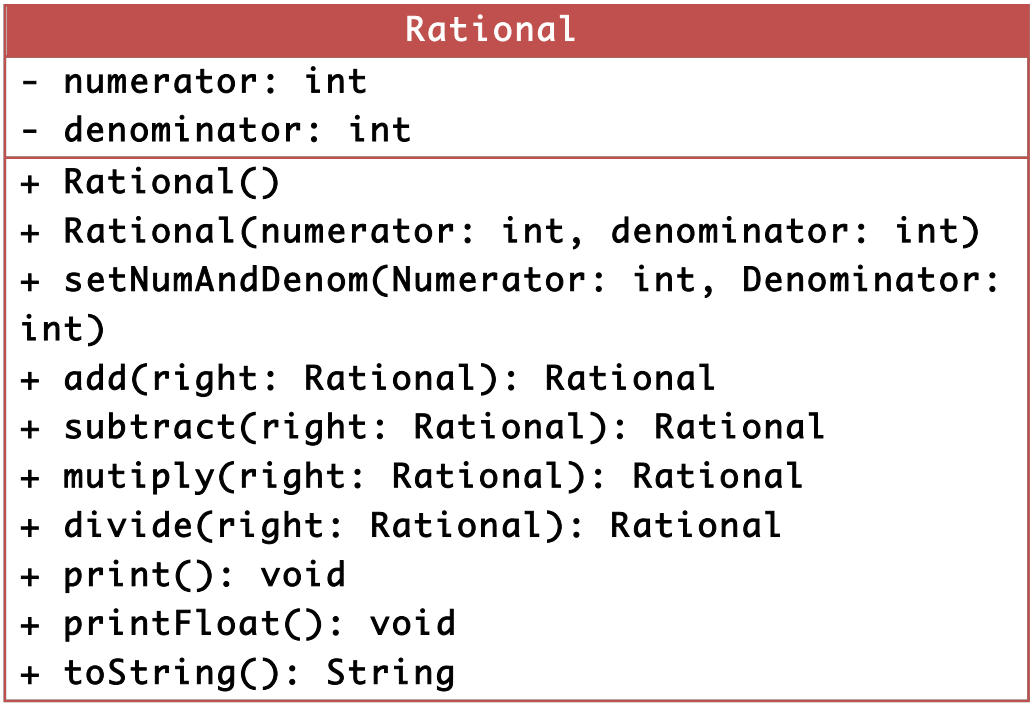
- Create a rational number $\frac{1}{2}$.
- After that create a rational number $\frac{2}{3}$.
- Print the following both as a fraction and as a floating point:
  - $\frac{1}{2} + \frac{2}{3}$
  - $\frac{2}{3} - \frac{1}{2}$
  - $\frac{1}{2} \times \frac{2}{3}$
  - $\frac{1}{2} \div \frac{2}{3}$

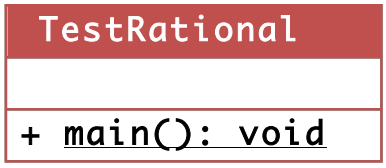
SAMPLE RUN

```
7/6
1.1666666666666667
7/6
1.1666666666666667
```

```
7/6
1.1666666666666667
7/6
1.1666666666666667
```

Solution


Rational
- numerator: int
- denominator: int
+ Rational()
+ Rational(numerator: int, denominator: int)
+ setNumAndDenom(Numerator: int, Denominator: int)
+ add(right: Rational): Rational
+ subtract(right: Rational): Rational
+ mutiply(right: Rational): Rational
+ divide(right: Rational): Rational
+ print(): void
+ printFloat(): void
+ toString(): String


TestRational
+ main(): void

```java
public class Rational {
    private int numerator, denominator;

    // no-argument constructor
    public Rational() {
        this(1, 1);
    }

    // initialize numerator part and denominator part
    public Rational(int numerator, int denominator) {
        setNumAndDenom(numerator, denominator);
    }

    public void setNumAndDenom(int numerator, int denominator) {
        if (denominator != 0) {
            this.numerator = numerator;
            this.denominator = denominator;
        } else {
            System.out.println("Error: division by zero.");
            System.exit(1);
        }
    }

    // add two Rational numbers
    public Rational add(Rational right) {
        int resultDenominator = denominator * right.denominator;
        int resultNumerator = numerator * right.denominator + right.numerator
                * denominator;
        return new Rational(resultNumerator, resultDenominator);
    }

    // subtract two Rational numbers
    public Rational subtract(Rational right) {
        int resultDenominator = denominator * right.denominator;
        int resultNumerator = numerator * right.denominator - right.numerator
                * denominator;
        return new Rational(resultNumerator, resultDenominator);
    }

    // multiply two Rational numbers
    public Rational multiply(Rational right) {
        return new Rational(numerator * right.numerator, denominator
                * right.denominator);
    }
```

```
    // divide two Rational numbers
    public Rational divide(Rational right) {
        return new Rational(numerator * right.denominator, denominator
                * right.numerator);

    }

    // return String representation of a Rational number
    public String toString() {
        return numerator + "/" + denominator;
    }

    public void print(){
        System.out.println(toString());
    }

    public void printFloat() {
        System.out.println((double) numerator / denominator);
    }
} // end class Rational
```

```
public class TestRational {
    public static void main(String[] args) {
        Rational n1 = new Rational(1, 2);
        Rational n2 = new Rational(2, 3);
        Rational addResult = n1.add(n2);
        addResult.print();
        addResult.printFloat();
        Rational subResult = n1.add(n2);
        subResult.print();
        subResult.printFloat();
        Rational mulResult = n1.add(n2);
        mulResult.print();
        mulResult.printFloat();
        Rational divResult = n1.add(n2);
        divResult.print();
        divResult.printFloat();
    }

}
```

Exercise 3

* Change class `Rational` as following:

- Add the following two method:
  - `reduce()` a *private* method that reduces the current rational number to most basic form. Reduction is done by dividing both the numerator and denominator by the greatest common devisor (gcd) of the two numbers. For example,

    $$\gcd(2, 4) = 2$$

    which means that, to reduce $\frac{2}{4}$, we divide 2 by 2 and 4 by 2. Doing so we will get $\frac{1}{2}$.
  - `equals(Rational otherObject)` a *public* method which returns true if the current rational number is equivalent to rational number `otherObject`. (*Hint*: the two rational numbers should be reduced before checking equivalency. For example, the fraction $\frac{2}{4}$ is equivalent to the reduced form $\frac{1}{2}$). Objects of type `Rational` must always store the fraction in reduced form such that user of class `Rational` does not have to call method reduce (i.e., it is a private method).
- Change method `printFloat()` by providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point (*Hint*: use class `java.text.DecimalFormat`. Check its javadoc online to understand how it works).

Solution

UML for new class after changes:

Rational
- numerator: int - denominator: int
+ Rational() + Rational(numerator: int, denominator: int) + setNumAndDenom(Numerator: int, Denominator: int) + add(right: Rational): Rational + subtract(right: Rational): Rational + mutiply(right: Rational): Rational + divide(right: Rational): Rational + print(): void + printFloat(): void + toString(): String + equals(otherObject: Rational): Boolean - reduce(): void

Here are code changes to previous class code:

1- Change setter method to call reduce():

```
public void setNumAndDenom(int numerator, int denominator) {
    if (denominator != 0) {
        this.numerator = numerator;
        this.denominator = denominator;
        reduce();
    } else {
        System.out.println("Error: division by zero.");
        System.exit(1);
    }
}
```

2- Add methods equals() and reduce():

```
public boolean equals(Rational otherObject){
    return numerator == otherObject.numerator && denominator == otherObject.denominator;
}

// reduce the fraction
private void reduce() {
    int gcd = 0;
    int smaller;
    if (numerator < denominator)

        smaller = numerator;
    else
        smaller = denominator;
    for (int divisor = smaller; divisor >= 2; divisor--) {
        if (numerator % divisor == 0 && denominator % divisor == 0) {
            gcd = divisor;
            break;

        }
    }
    if (gcd != 0) {
        numerator /= gcd;
        denominator /= gcd;
    }
}
```

Question: Why did not we call method `reduce` in method `equals`?

3- Change method printFloat() as following:

```
public void printFloat(int digits) {
    String format = "0.";

    // get format string
    for (int i = 0; i < digits; i++){
        format += "0";
    }

    java.text.DecimalFormat twoDigits = new java.text.DecimalFormat(format);

    System.out.println(twoDigits.format((double) numerator / denominator).toString());
}
```

Done…