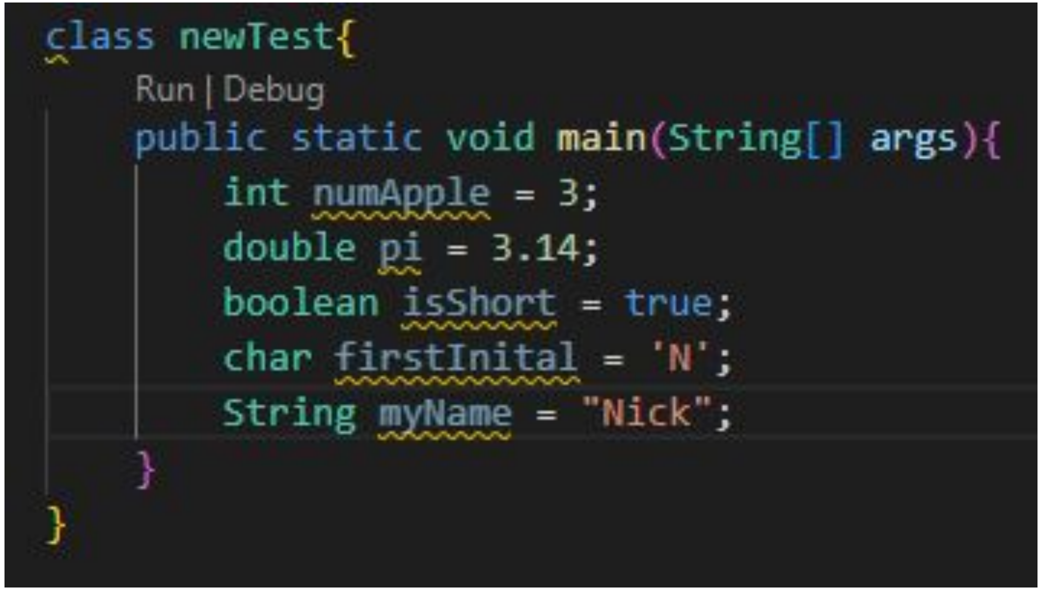
COSC:101 Midterm Review

1.3 Variables and Data Types

- Variables- allow us to store information such as numbers, words, or true/false expressions
    - Variables are composed of: a name, type, and value

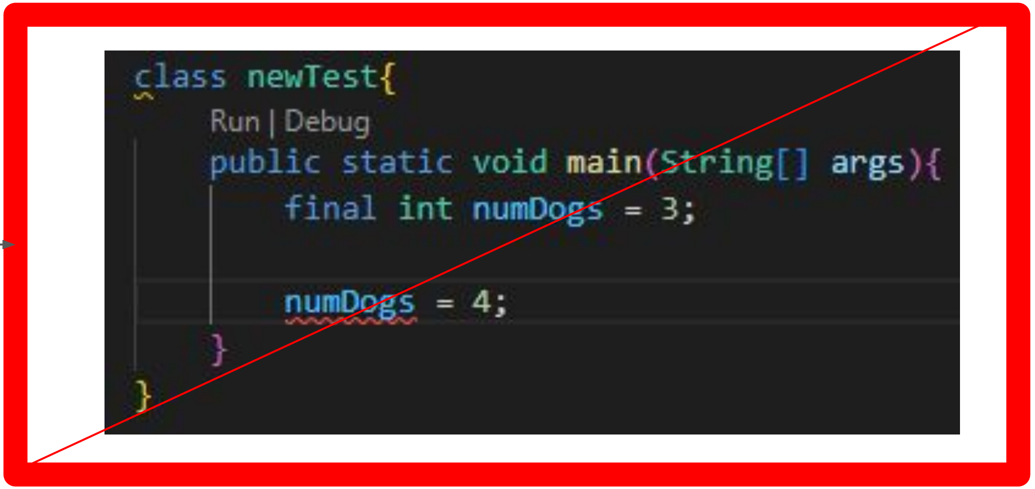


```
class newTest{
    Run | Debug
    public static void main(String[] args){
        int numApple = 3;
        double pi = 3.14;
        boolean isShort = true;
        char firstInital = 'N';
        String myName = "Nick";
    }
}
```

Cont.

- Primitive vs reference types
  - Primitive types: int (whole number), double (can have decimals), char (character), boolean (true or false)
  - Reference types: store address that points to where value is in memory (ie String)
- Final keyword- not allowed to change the value later on (will get error if try to change it)

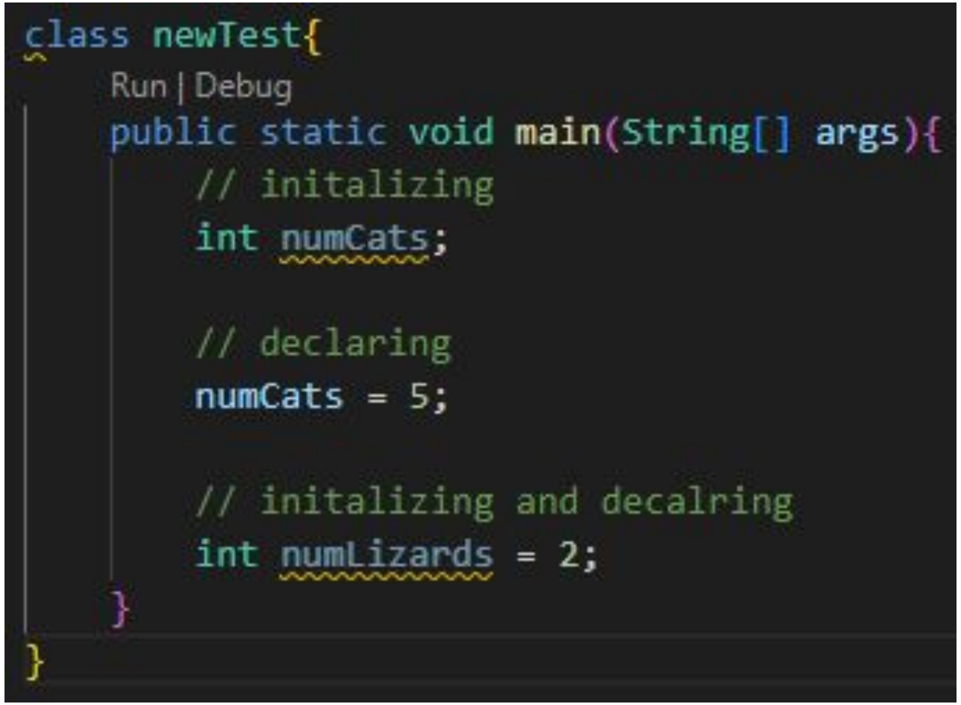
Will give you an error



```
class newTest{
    Run | Debug
    public static void main(String[] args){
        final int numDogs = 3;

        numDogs = 4;
    }
}
```

Cont.

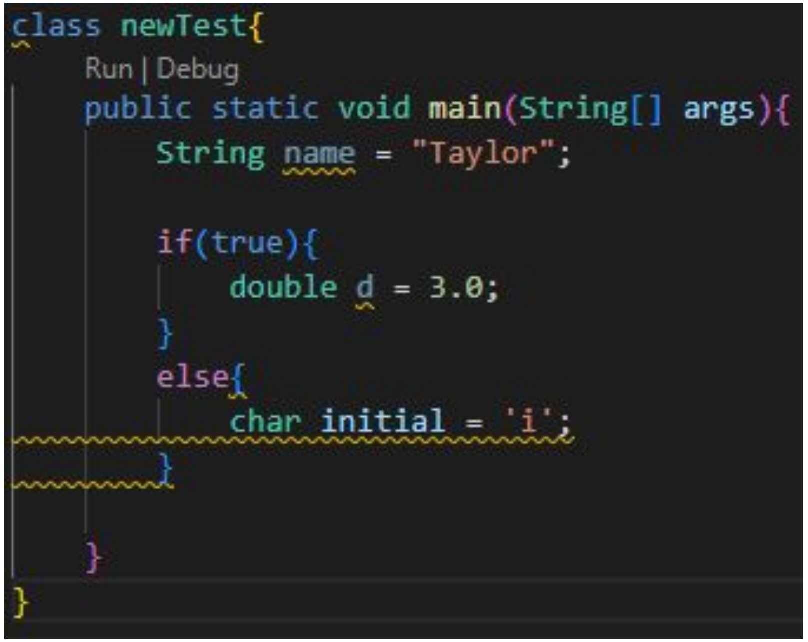
- Declaring vs. initializing a variable
    - Declaring: creating it (data type and name)
    - Initializing: giving it its initial value



```
class newTest{
    Run | Debug
    public static void main(String[] args){
        // initalizing
        int numCats;

        // declaring
        numCats = 5;

        // initalizing and decalring
        int numLizards = 2;
    }
}
```

Scope

- Variables that are created within { } are only "alive" within those braces



```
class newTest{
    Run | Debug
    public static void main(String[] args){
        String name = "Taylor";

        if(true){
            double d = 3.0;
        }
        else{
            char initial = 'i';
        }

    }
}
```

1.4 Expression and Assignment Statements

- Arithmetic expressions
    - + : addition operator
    - - : subtraction operator
    - * : multiplication operator
    - / : division operator
    - % : modulus operator
- Remember PEMDAS!

Mod review (examples):

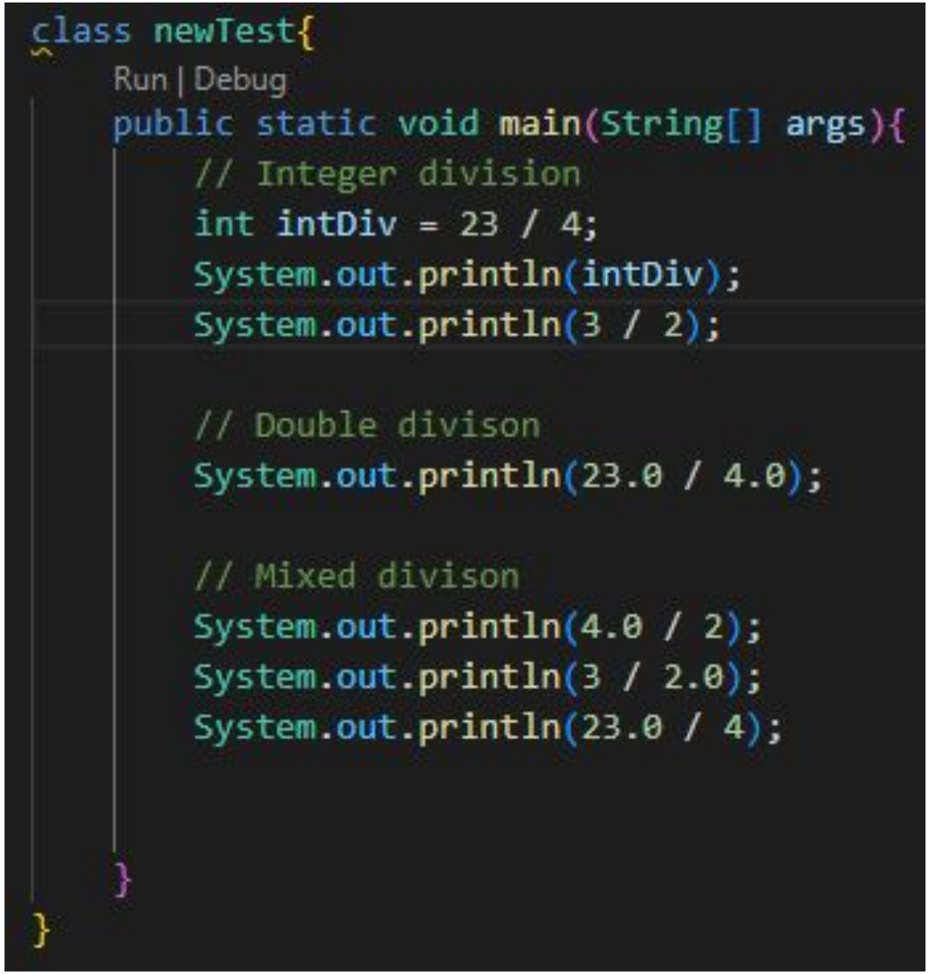
10 / 2 = 5
10 % 2 = 0

10 / 3 = 3
10 % 3 = 1

14 / 5 = 2
14 % 5 = 4

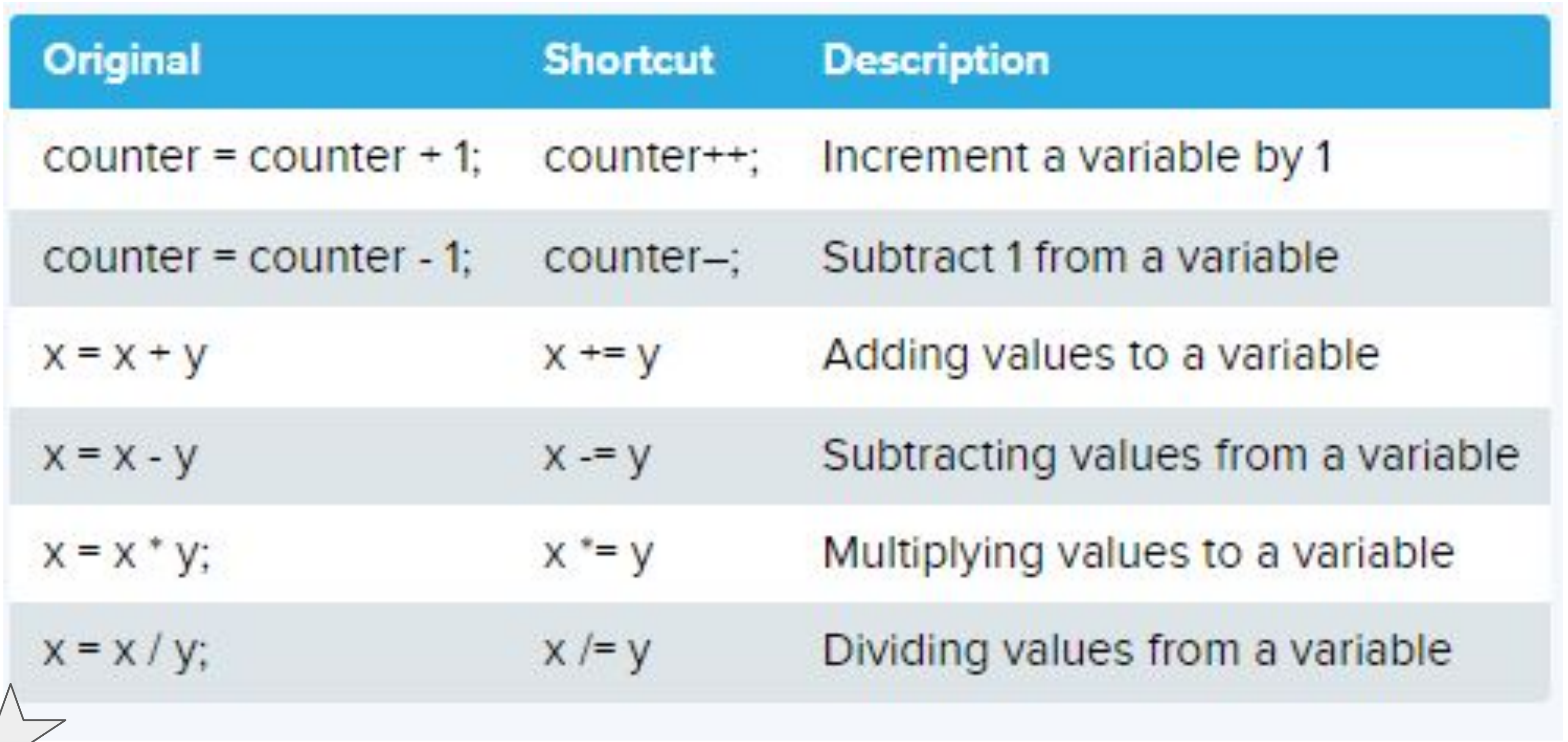
Cont.

- Integer division vs. double division vs. mixed



```
class newTest{
    Run | Debug
    public static void main(String[] args){
        // Integer division
        int intDiv = 23 / 4;
        System.out.println(intDiv);
        System.out.println(3 / 2);

        // Double divison
        System.out.println(23.0 / 4.0);

        // Mixed divison
        System.out.println(4.0 / 2);
        System.out.println(3 / 2.0);
        System.out.println(23.0 / 4);


    }
}
```

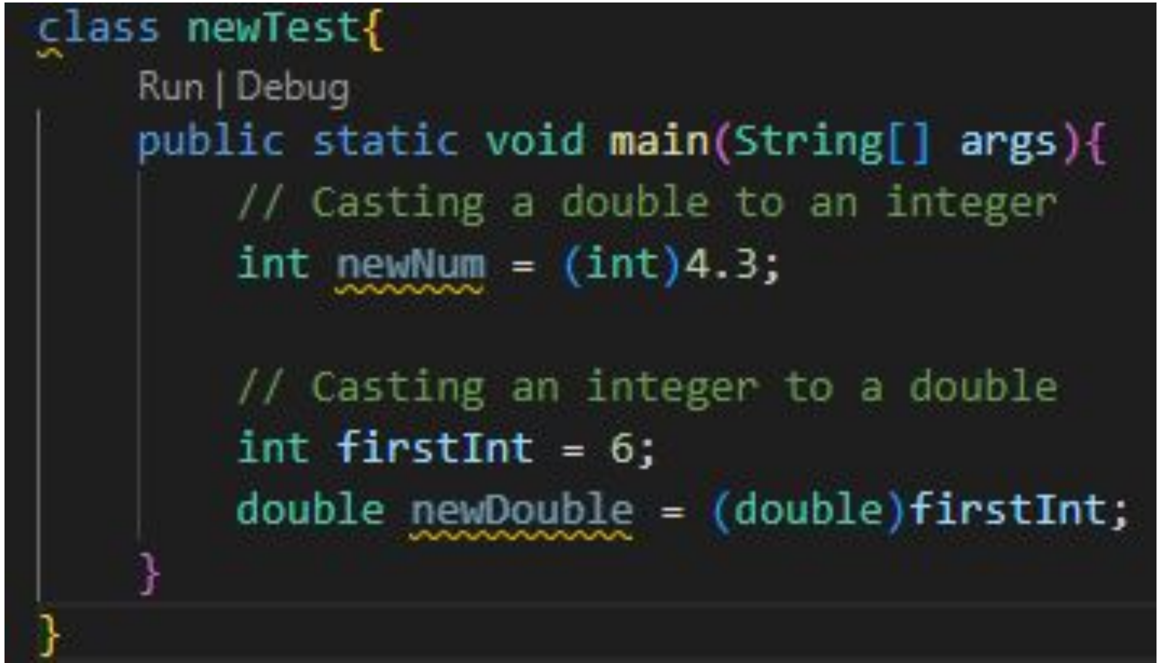
1.5 Compound Assignment Operators



Original	Shortcut	Description
counter = counter + 1;	counter++;	Increment a variable by 1
counter = counter - 1;	counter–;	Subtract 1 from a variable
x = x + y	x += y	Adding values to a variable
x = x - y	x -= y	Subtracting values from a variable
x = x * y;	x *= y	Multiplying values to a variable
x = x / y;	x /= y	Dividing values from a variable

1.6 Casting and Range of Variables

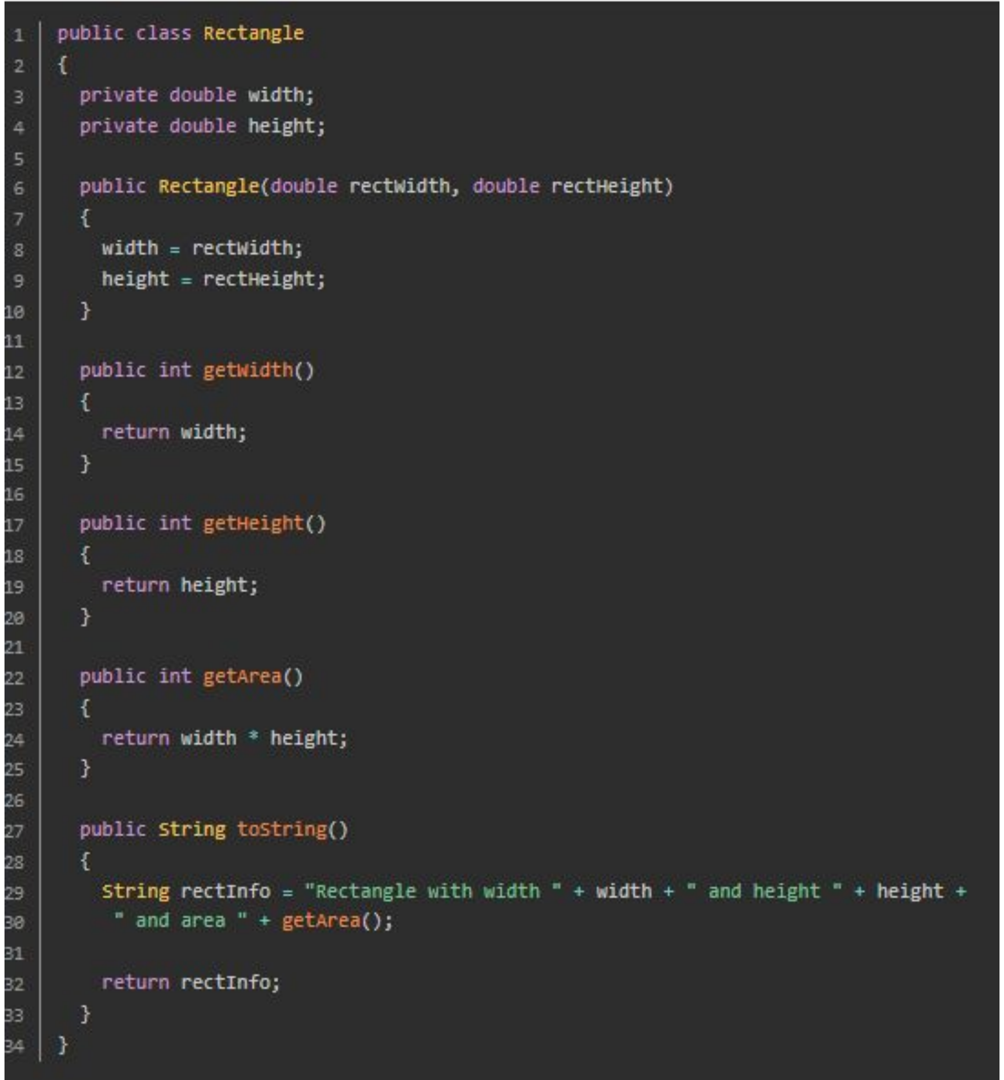
- Casting - changing data types
- Int to double and double to int



```
class newTest{
    Run | Debug
    public static void main(String[] args){
        // Casting a double to an integer
        int newNum = (int)4.3;

        // Casting an integer to a double
        int firstInt = 6;
        double newDouble = (double)firstInt;
    }
}
```

2.1 Objects: Instances of Classes

- Objects- variables of user-defined data types
- Have a state (contains info about the object) and behavior (actions that can be performed on the object)
- Classes- the templates you use for creating objects (the blueprint)
- Instance- a specific version of an object that can differ in numerous ways



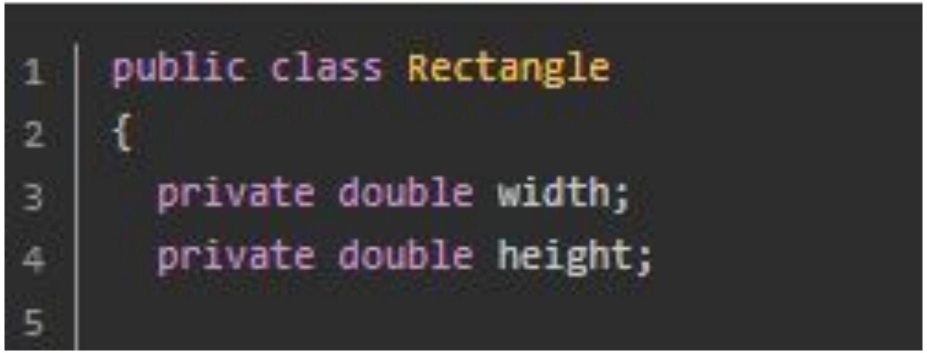
```java
public class Rectangle
{
  private double width;
  private double height;

  public Rectangle(double rectWidth, double rectHeight)
  {
    width = rectWidth;
    height = rectHeight;
  }

  public int getWidth()
  {
    return width;
  }

  public int getHeight()
  {
    return height;
  }

  public int getArea()
  {
    return width * height;
  }

  public String toString()
  {
    String rectInfo = "Rectangle with width " + width + " and height " + height +
     " and area " + getArea();

    return rectInfo;
  }
}
```

2.2 Creating and Storing Objects (Instantiation)

Every object is created using the keyword new followed by a call to the class' constructor

3 main parts of a class:

1. Instance variables (attributes) - state of object
    - Not usually initialized when declared

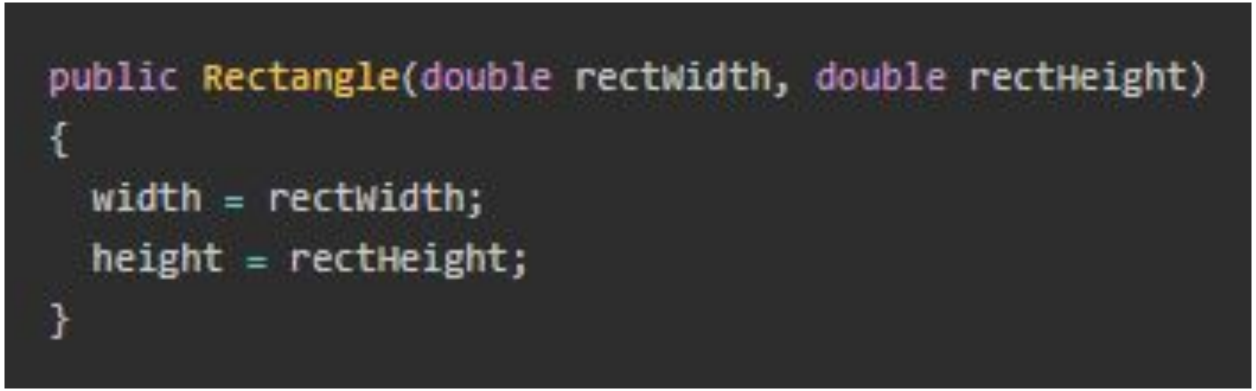


```java
public class Rectangle
{
  private double width;
  private double height;

```

Cont.

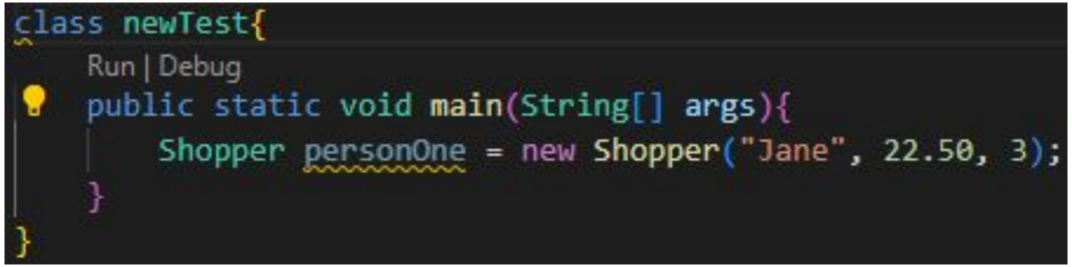
2. Constructor- used to instantiate (create an instance of) an object

- Has the same name as the class, is always declared public, and has no return type
- Signature- the constructor name and the parameter list
- Parameter list- lists the types of the values that are passed in and their variable names
- Argument is a value that is passed into a constructor (or any method) when called

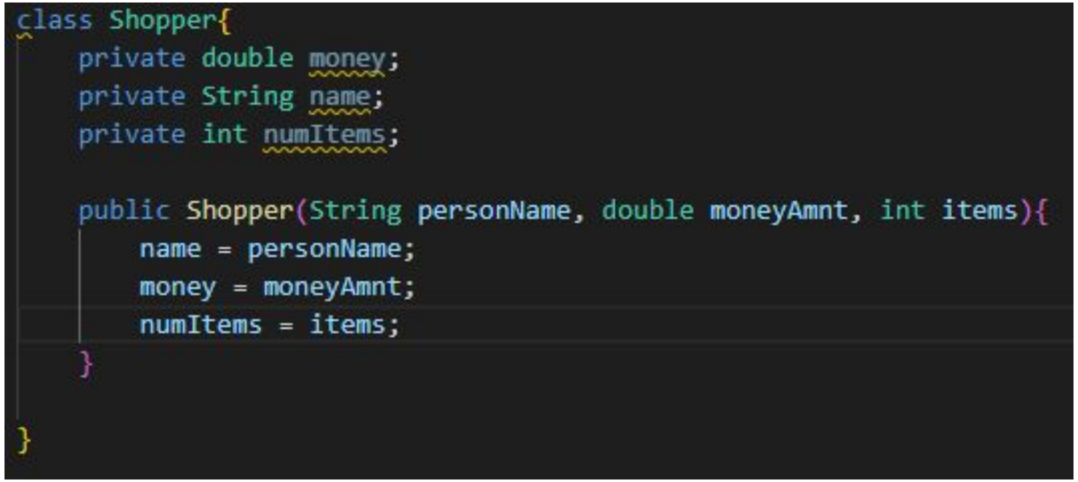


```
public Rectangle(double rectWidth, double rectHeight)
{
  width = rectWidth;
  height = rectHeight;
}
```

Example of Calling the Constructor



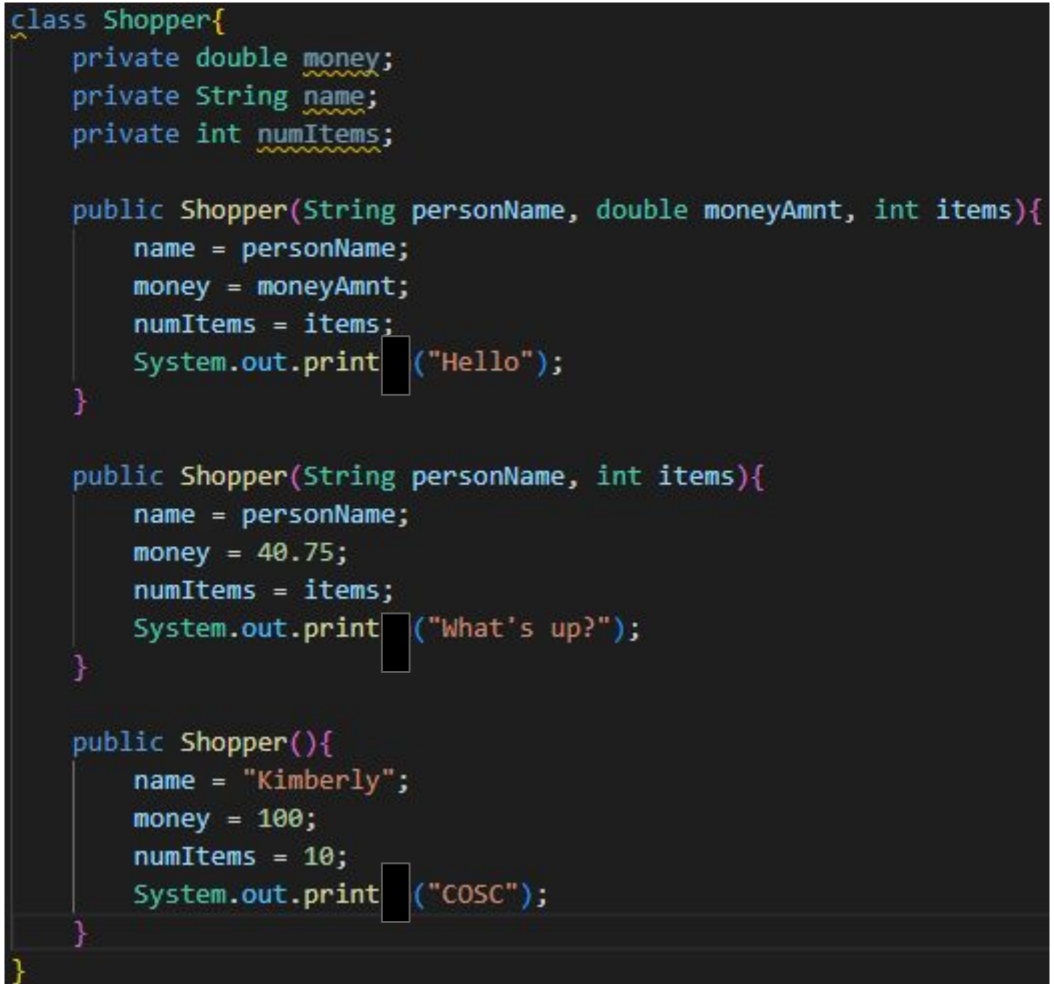
```
class newTest{
    Run | Debug
    public static void main(String[] args){
        Shopper personOne = new Shopper("Jane", 22.50, 3);
    }
}
```



```
class Shopper{
    private double money;
    private String name;
    private int numItems;

    public Shopper(String personName, double moneyAmnt, int items){
        name = personName;
        money = moneyAmnt;
        numItems = items;
    }

}
```

Cont.

- Overloading constructor

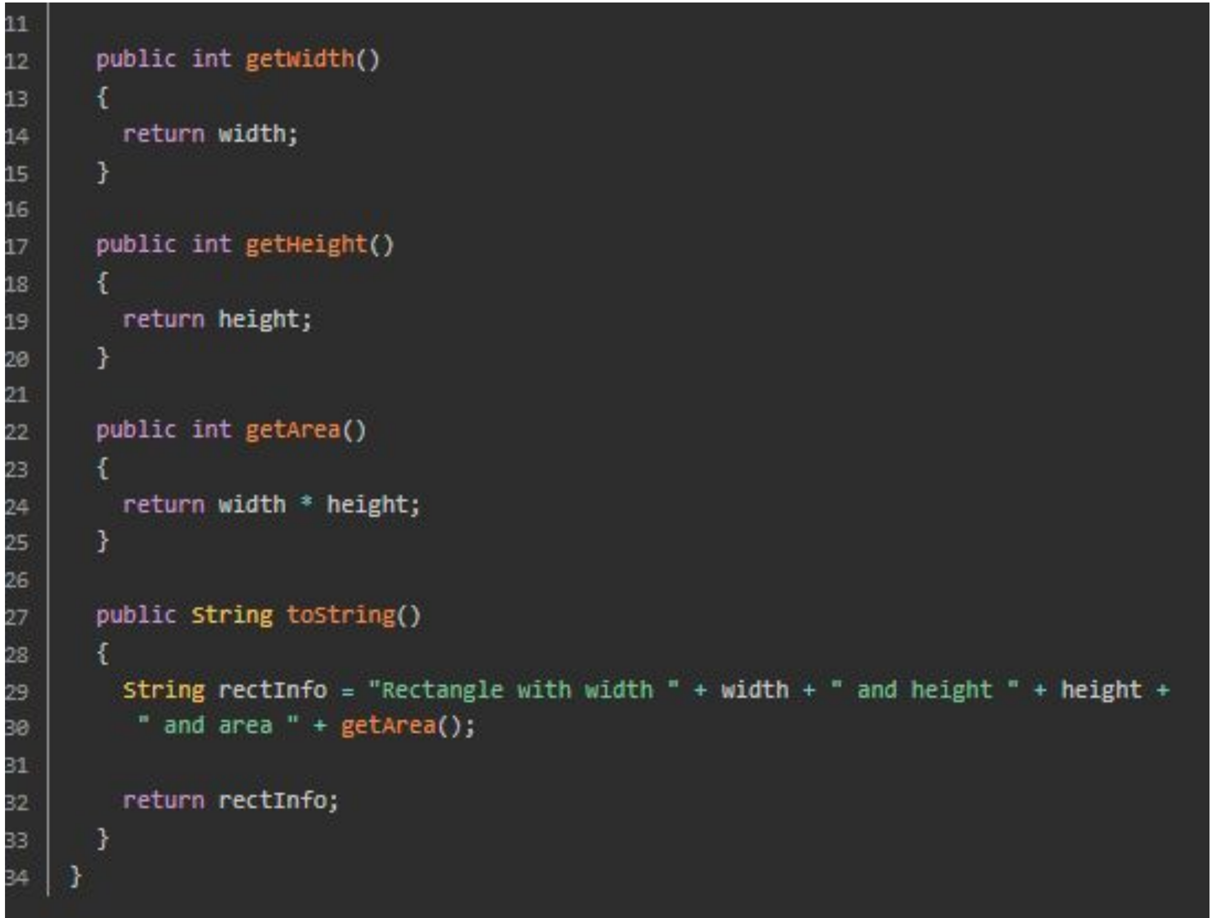


```java
class Shopper{
    private double money;
    private String name;
    private int numItems;

    public Shopper(String personName, double moneyAmnt, int items){
        name = personName;
        money = moneyAmnt;
        numItems = items;
        System.out.print[redacted]("Hello");
    }

    public Shopper(String personName, int items){
        name = personName;
        money = 40.75;
        numItems = items;
        System.out.print[redacted]("What's up?");
    }

    public Shopper(){
        name = "Kimberly";
        money = 100;
        numItems = 10;
        System.out.print[redacted]("COSC");
    }
}
```

Cont.

3. Methods

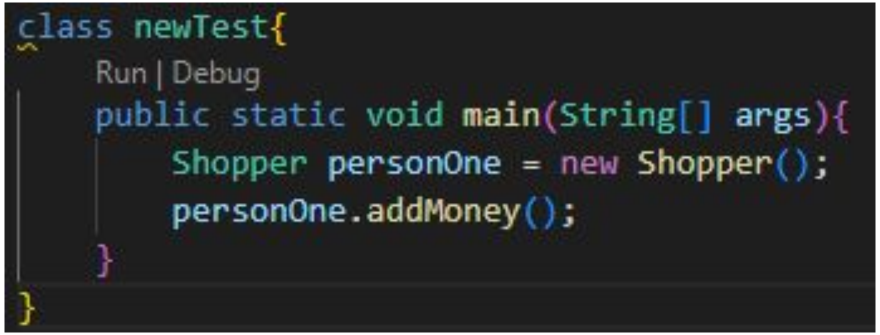
The behaviors of the object



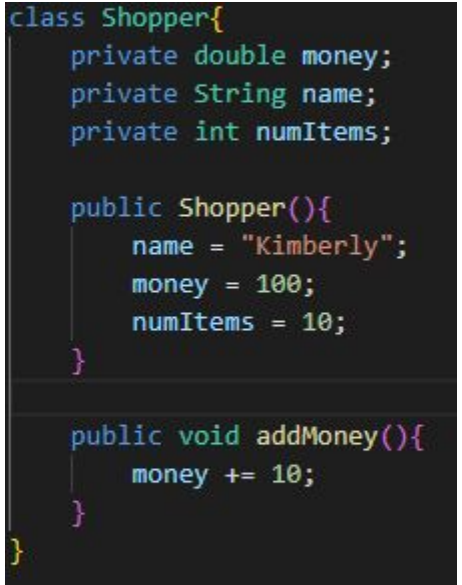
```
  public int getWidth()
  {
    return width;
  }

  public int getHeight()
  {
    return height;
  }

  public int getArea()
  {
    return width * height;
  }

  public String toString()
  {
    String rectInfo = "Rectangle with width " + width + " and height " + height +
     " and area " + getArea();

    return rectInfo;
  }
}
```

2.3 Calling a Void Method

- Methods- procedures that allow us to control and define the behavior of an object
  - Parts of a method: public/private, void, methodName, (), {}



```
class newTest{
    Run | Debug
    public static void main(String[] args){
        Shopper personOne = new Shopper();
        personOne.addMoney();
    }
}
```



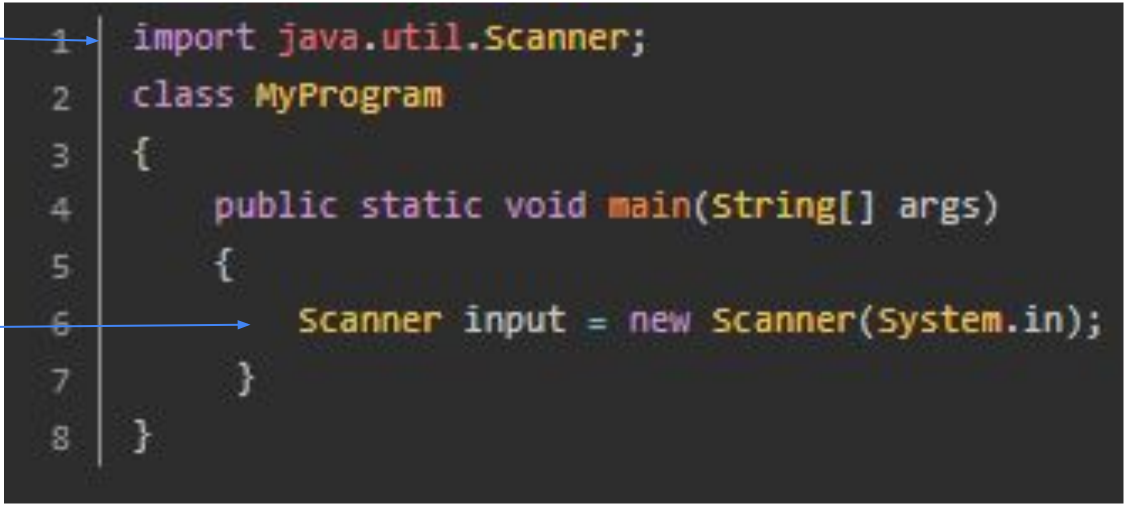
```
class Shopper{
    private double money;
    private String name;
    private int numItems;

    public Shopper(){
        name = "Kimberly";
        money = 100;
        numItems = 10;
    }

    public void addMoney(){
        money += 10;
    }
}
```

Cont.: Scanner

- A scanner object- a type of variable that allows you to read from the console
- Declare:
- Read in different types:

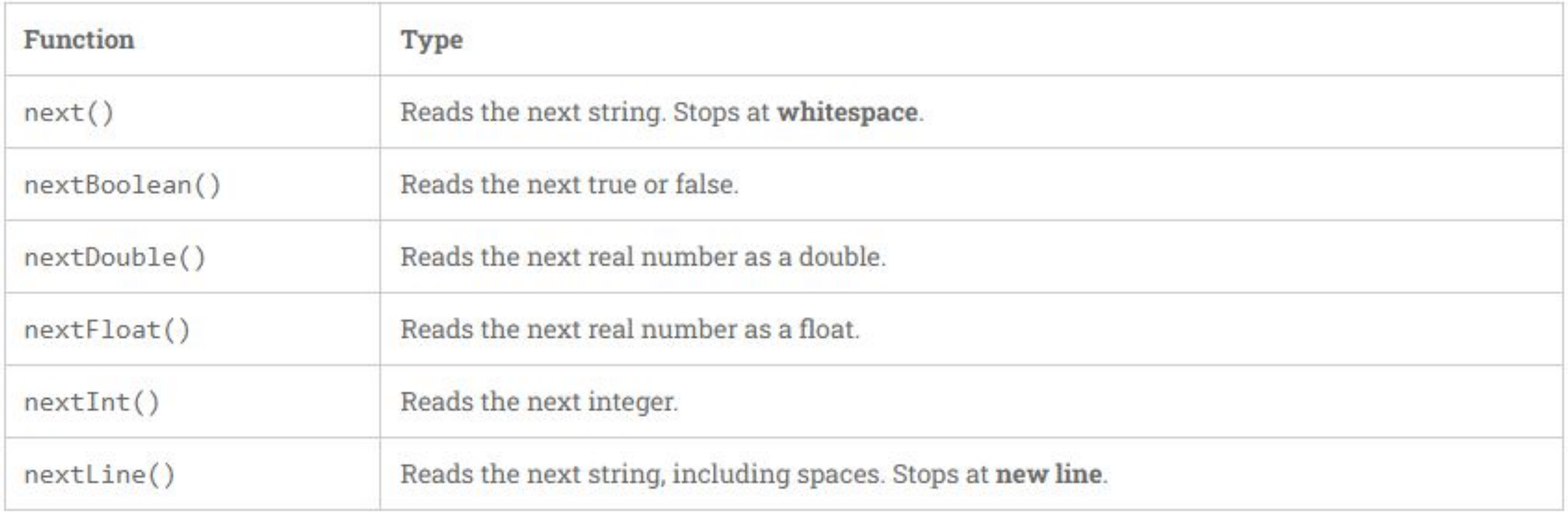
1. Import

2. Create input variable



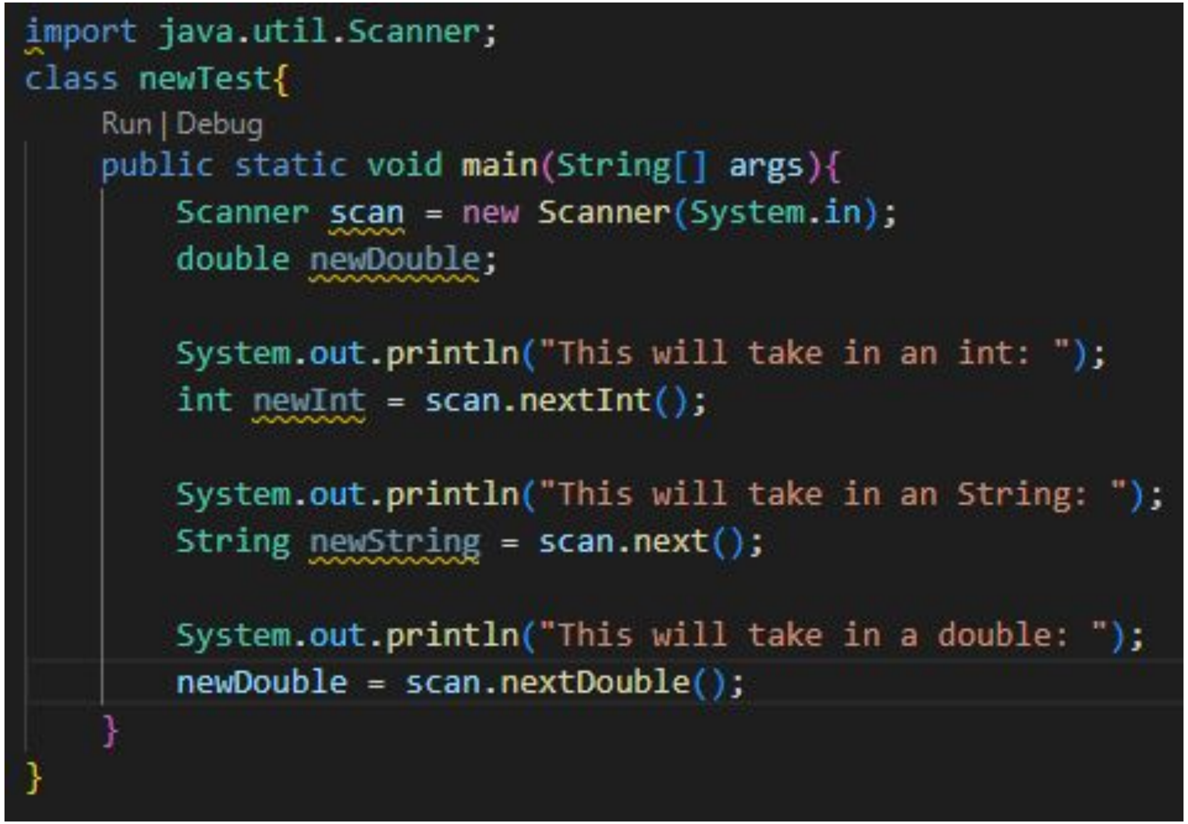
```
import java.util.Scanner;
class MyProgram
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
     }
}
```

Cont.: Scanner



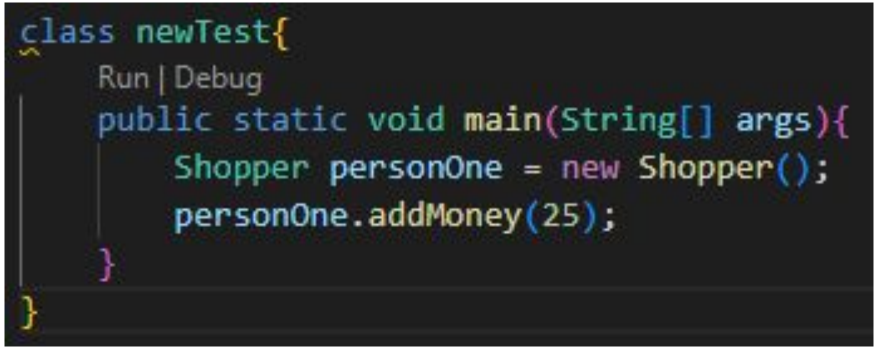
Function	**Type**
`next()`	Reads the next string. Stops at **whitespace**.
`nextBoolean()`	Reads the next true or false.
`nextDouble()`	Reads the next real number as a double.
`nextFloat()`	Reads the next real number as a float.
`nextInt()`	Reads the next integer.
`nextLine()`	Reads the next string, including spaces. Stops at **new line**.

Cont: Scanner

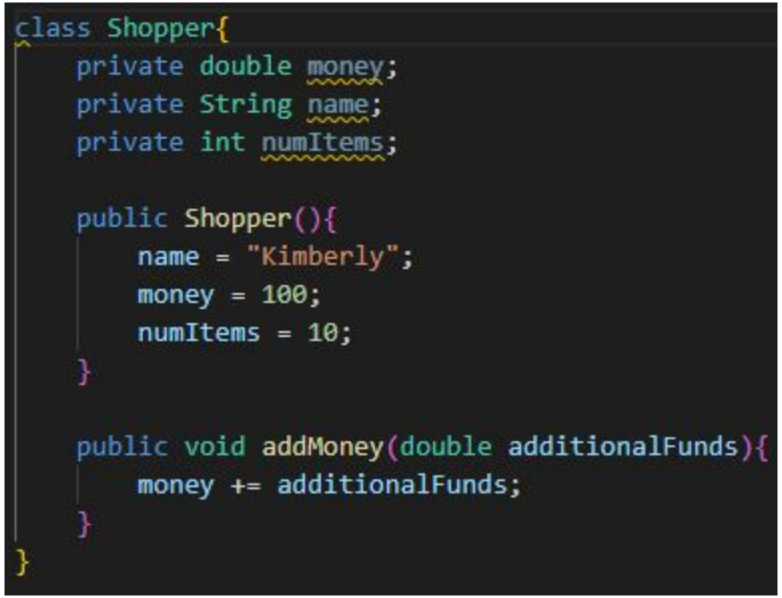

```java
import java.util.Scanner;
class newTest{
    Run | Debug
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        double newDouble;

        System.out.println("This will take in an int: ");
        int newInt = scan.nextInt();

        System.out.println("This will take in an String: ");
        String newString = scan.next();

        System.out.println("This will take in a double: ");
        newDouble = scan.nextDouble();
    }
}
```

2.4 Calling a Void Method with Parameters

- Pass by value- creating a copy to go through the method
- Method overload- same name, different ordered list (different number of parameters or types of parameters)



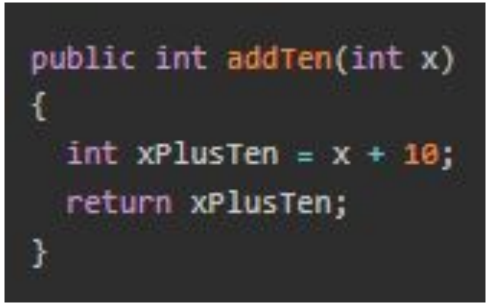
```java
class newTest{
    Run | Debug
    public static void main(String[] args){
        Shopper personOne = new Shopper();
        personOne.addMoney(25);
    }
}
```



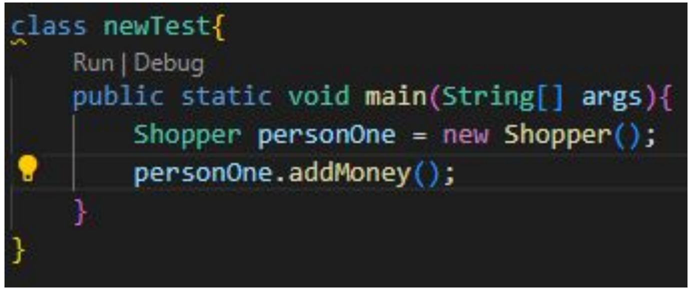
```java
class Shopper{
    private double money;
    private String name;
    private int numItems;

    public Shopper(){
        name = "Kimberly";
        money = 100;
        numItems = 10;
    }

    public void addMoney(double additionalFunds){
        money += additionalFunds;
    }
}
```

2.5 Calling a Non-void Method

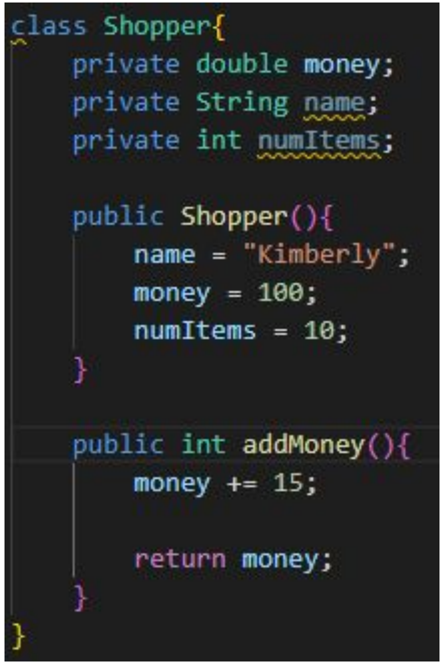
- Return statement- will allow you to pass a value back out of the method



```java
public int addTen(int x)
{
  int xPlusTen = x + 10;
  return xPlusTen;
}
```

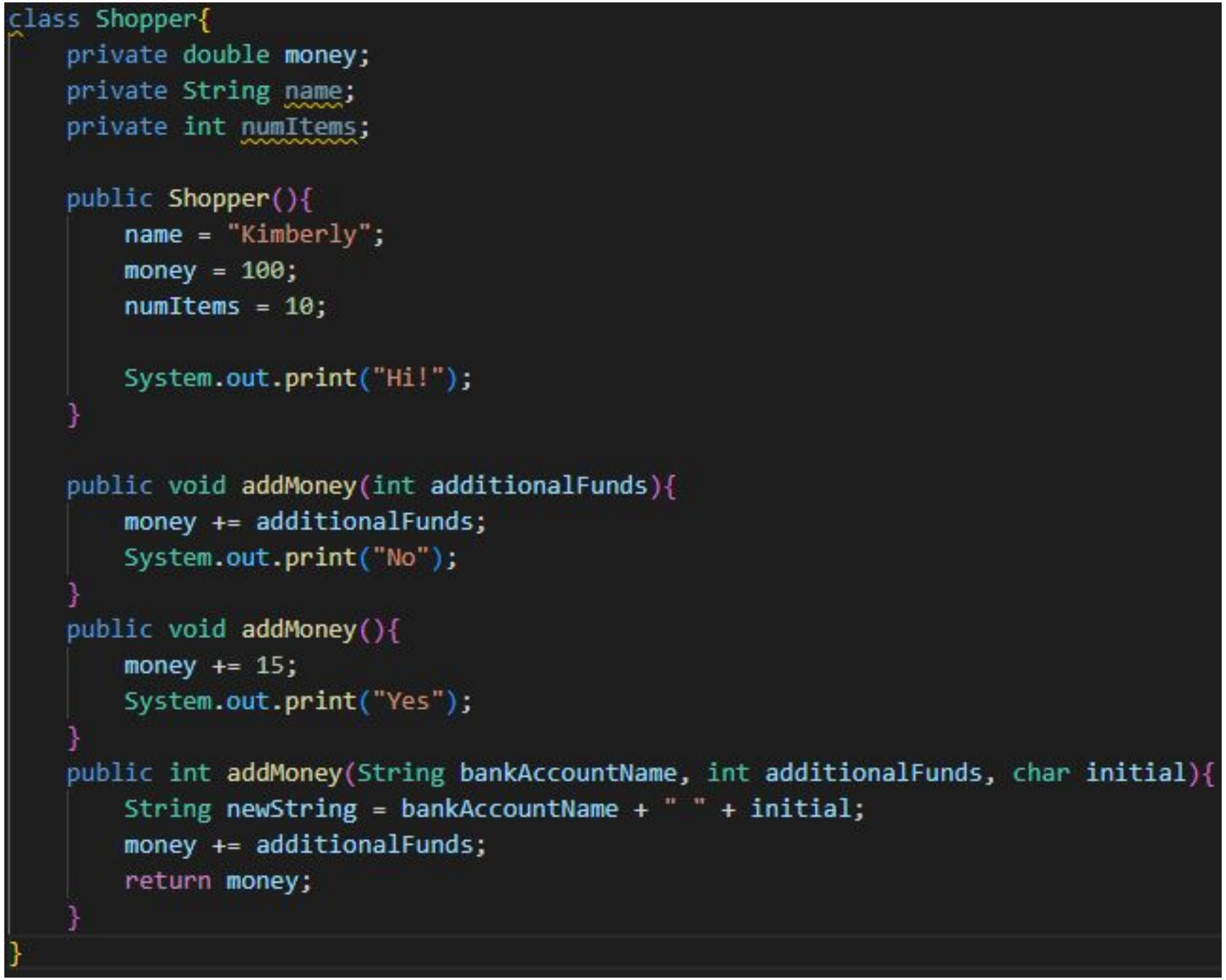


```java
class newTest{
    Run | Debug
    public static void main(String[] args){
        Shopper personOne = new Shopper();
        personOne.addMoney();
    }
}
```



```java
class Shopper{
    private double money;
    private String name;
    private int numItems;

    public Shopper(){
        name = "Kimberly";
        money = 100;
        numItems = 10;
    }

    public int addMoney(){
        money += 15;

        return money;
    }
}
```

Example of Method Overload



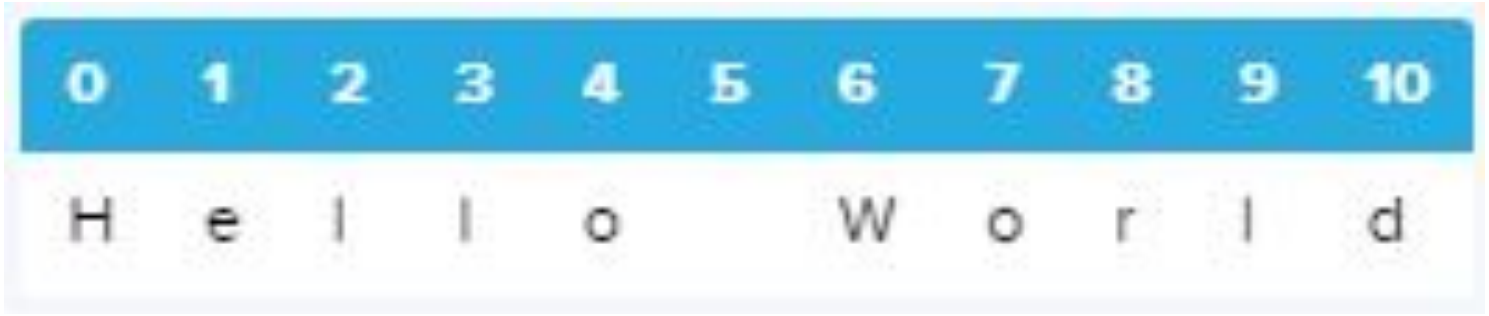
```java
class Shopper{
    private double money;
    private String name;
    private int numItems;

    public Shopper(){
        name = "Kimberly";
        money = 100;
        numItems = 10;

        System.out.print("Hi!");
    }

    public void addMoney(int additionalFunds){
        money += additionalFunds;
        System.out.print("No");
    }
    public void addMoney(){
        money += 15;
        System.out.print("Yes");
    }
    public int addMoney(String bankAccountName, int additionalFunds, char initial){
        String newString = bankAccountName + " " + initial;
        money += additionalFunds;
        return money;
    }
}
```

2.6 String Objects: Concatenation, Literals, and More

- String literal- string inside double quotes
- String concatenation: use + to combine strings

2.7 String Methods

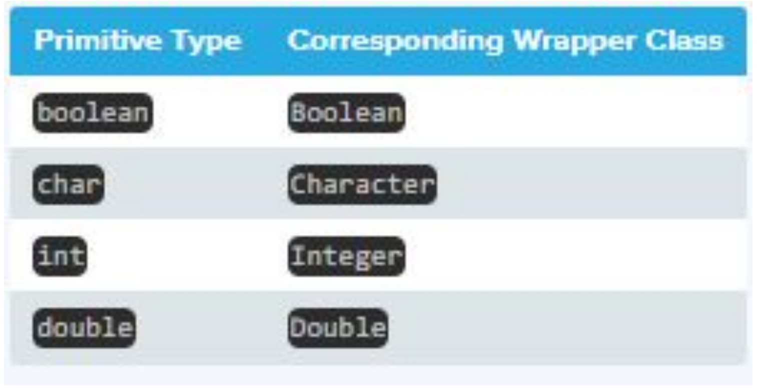
- Indexes
- substring, indexOf, length



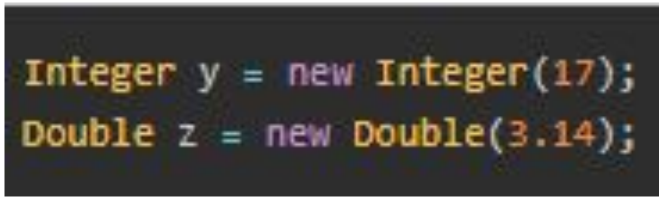
0	1	2	3	4	5	6	7	8	9	10
H	e	l	l	o		W	o	r	l	d

2.8 Wrapper Classes: Integers and Doubles

- Wrapper class- a class that contains or “wraps” primitive data types as an object
- Can take primitive data types and convert them into an object, provide static methods that allow you to perform some basic number operations, such as converting data from a string to a number.



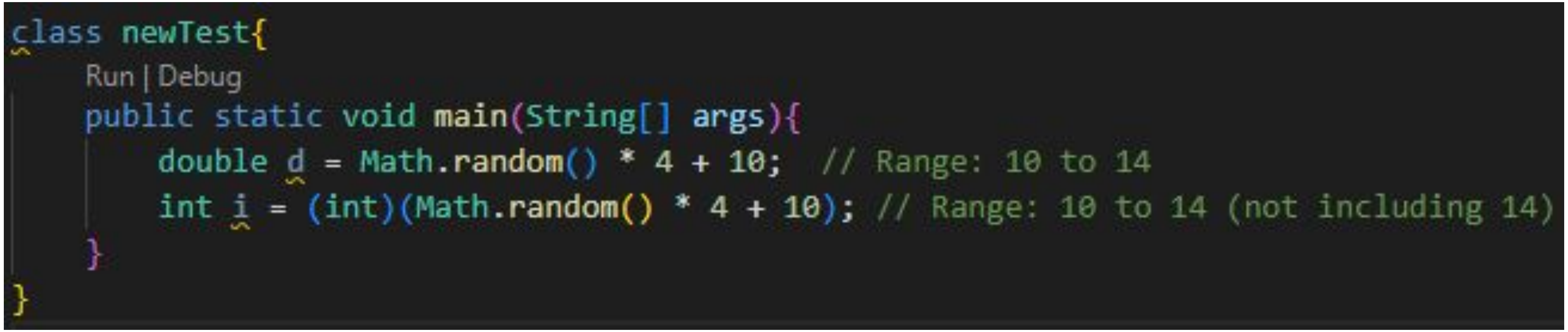
Primitive Type	Corresponding Wrapper Class
`boolean`	`Boolean`
`char`	`Character`
`int`	`Integer`
`double`	`Double`



```
Integer y = new Integer(17);
Double z = new Double(3.14);
```

2.9 Using the Math Class

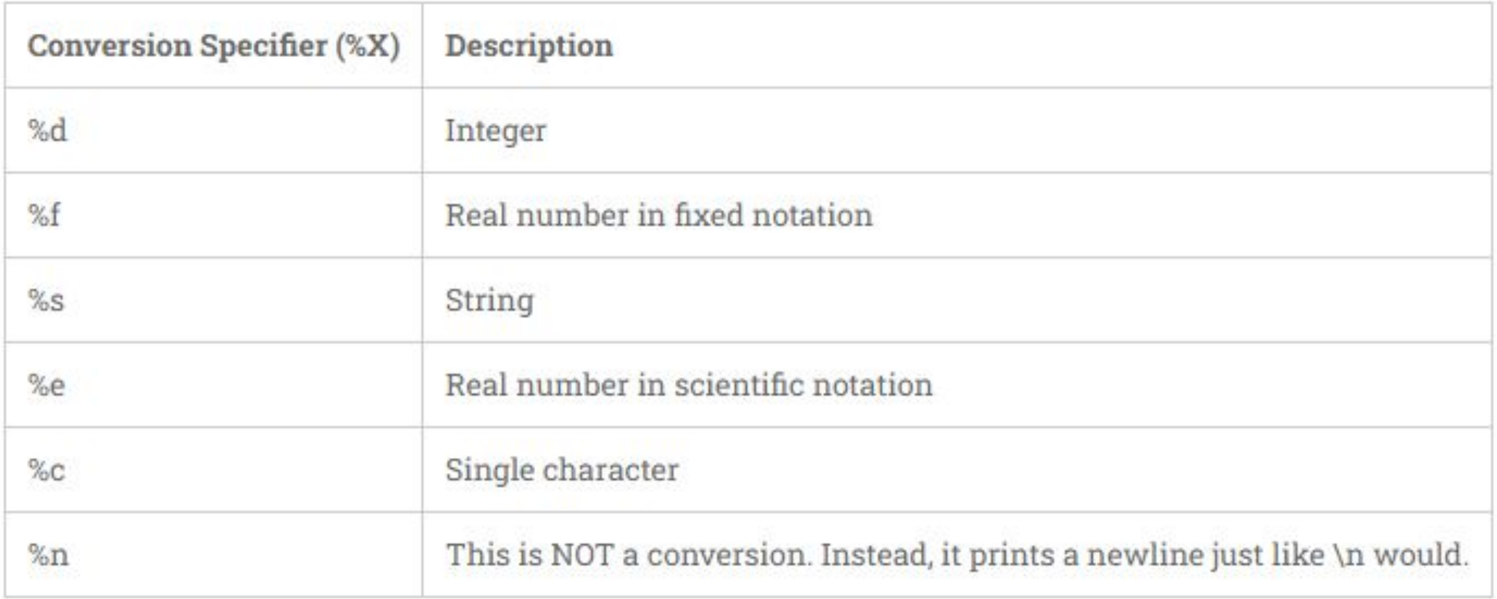
- Math.random()
  - Between 0 (inclusive) and 1 (exclusive)



```
class newTest{
    Run | Debug
    public static void main(String[] args){
        double d = Math.random() * 4 + 10;  // Range: 10 to 14
        int i = (int)(Math.random() * 4 + 10); // Range: 10 to 14 (not including 14)
    }
}
```

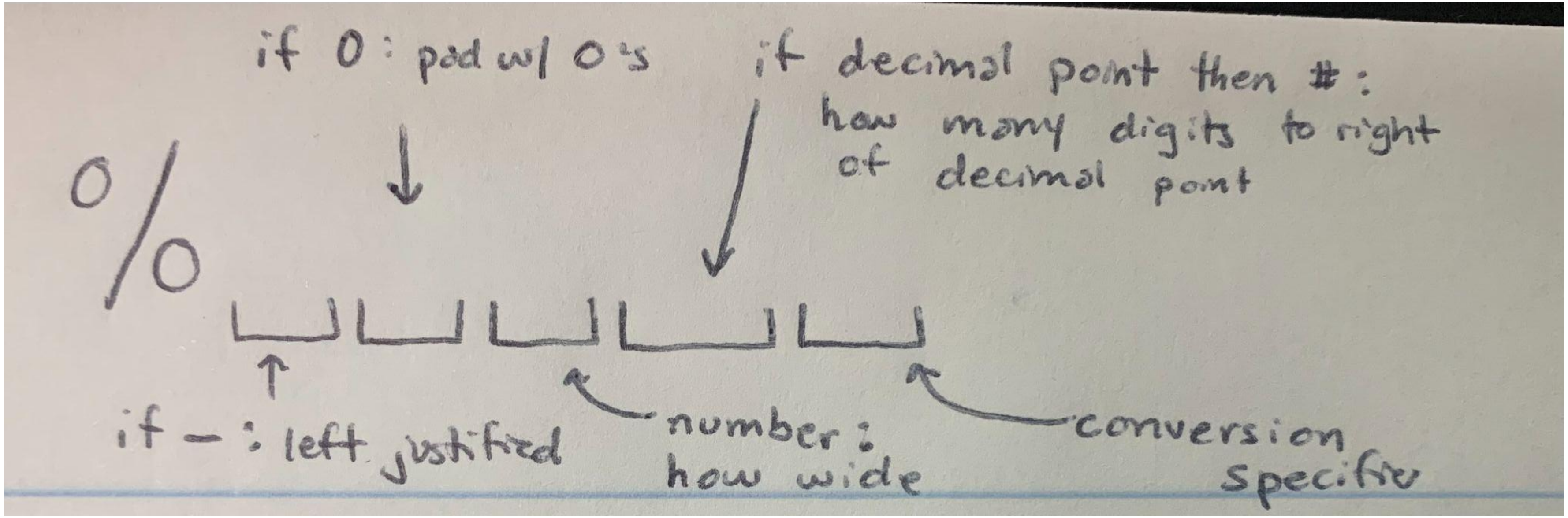
System.out.format

- Specifiers



Conversion Specifier (%X)	Description
%d	Integer
%f	Real number in fixed notation
%s	String
%e	Real number in scientific notation
%c	Single character
%n	This is NOT a conversion. Instead, it prints a newline just like \n would.

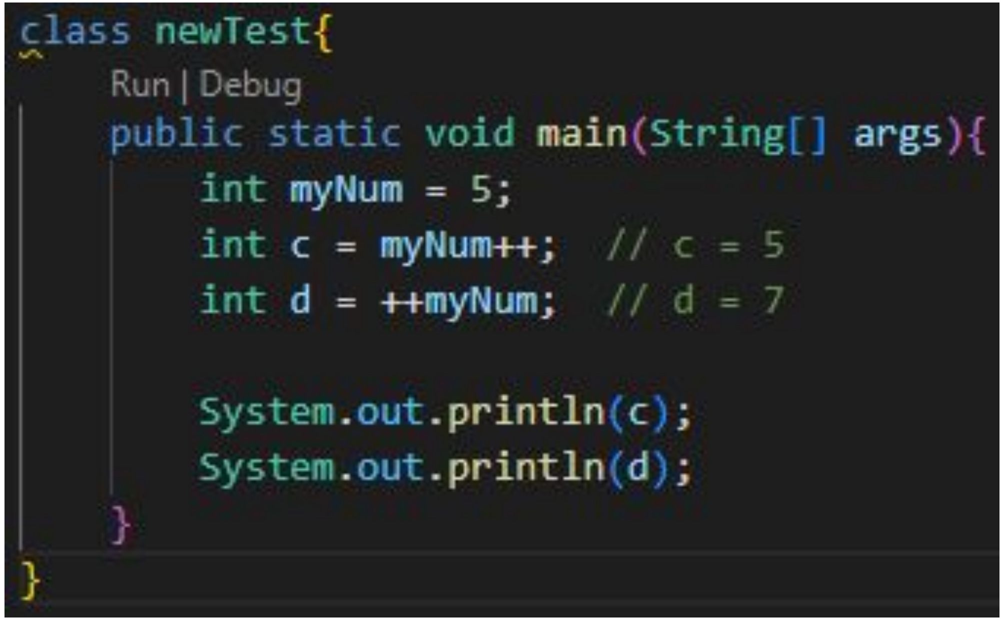
Cont.



if 0 : pad w/ 0's
if decimal point then # : how many digits to right of decimal point
%
if - : left justified
number : how wide
conversion specifier

Prefix vs postfix increment

- Prefix increment
  - ++myNum <- increment myNum FIRST
- Postfix increment
  - myNum++ <- increment myNum AFTER returning value to the variable



```
class newTest{
    Run | Debug
    public static void main(String[] args){
        int myNum = 5;
        int c = myNum++;  // c = 5
        int d = ++myNum;  // d = 7

        System.out.println(c);
        System.out.println(d);
    }
}
```

Questions?