System.out

frequently used methods

Name	Use
print(x)	print x and stay on the current line
println(x)	print x and move to next line down
printf(s,x)	print x according to s specifications

Escape Sequences

frequently used combinations

Name	Use
\t	tabs over five spaces
\n	moves to front of next line
\b	deletes previous character
\r	moves to front of current line
\\	nets one backslash \
\"	nets one double quote "
\’	nets one single quote ’

All Data Types

data type	memory usage	min .. max
byte	8 bits	-128 to 127
short	16 bits	-32768 to 32767
int	32 bits	-2 billion to 2 billion
long	64 bits	-big to +big
float	32 bits	-big to +big
double	64 bits	-big to +big
char	16 bit unsigned	0 - 65535
reference	32 bits	n/a

It is important to know all data types and what each one can store.

References/Objects

primitive	object
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Scanner

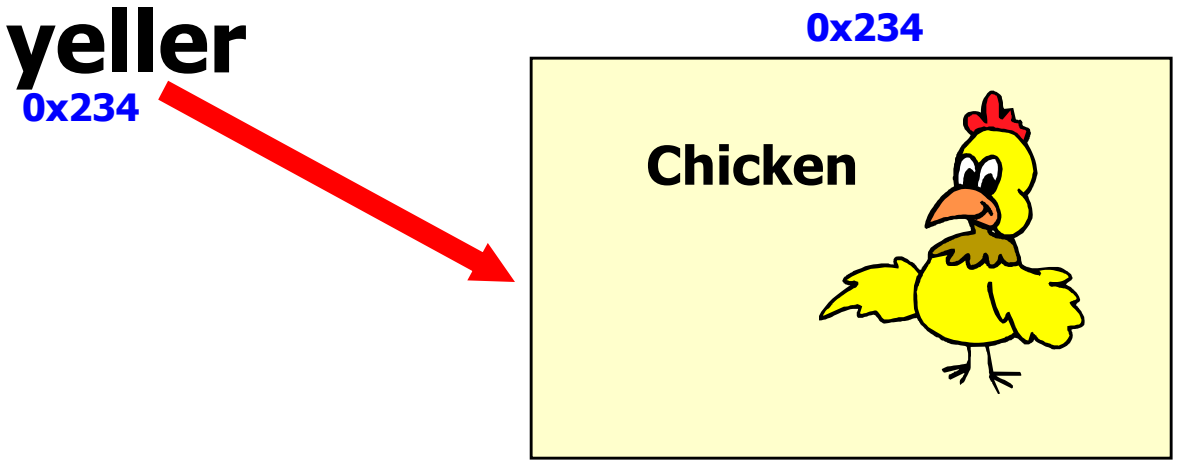
frequently used methods

Name	Use
nextInt()	returns the next int value
nextDouble()	returns the next double value
nextFloat()	returns the next float value
nextLong()	returns the next long value
nextByte()	returns the next byte value
nextShort()	returns the next short value
next()	returns the next one word String
nextLine()	returns the next multi word String

```
import  java.util.Scanner;
```

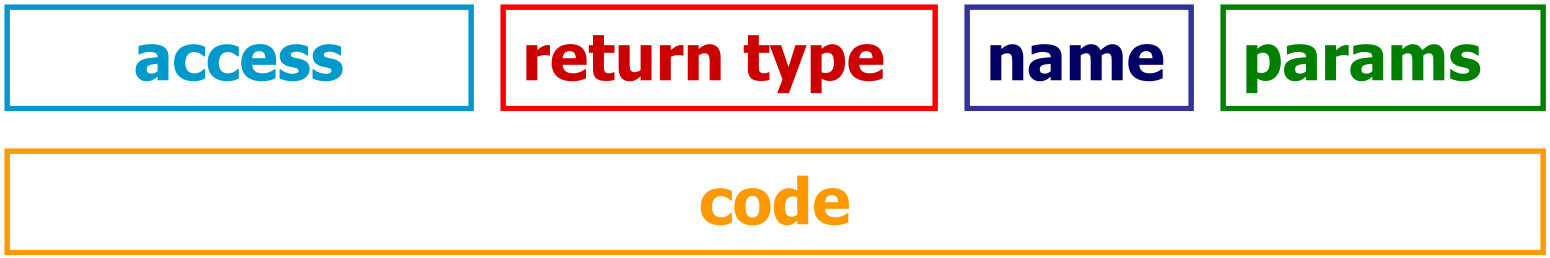
Object Instantiation

Chicken yeller = new Chicken();



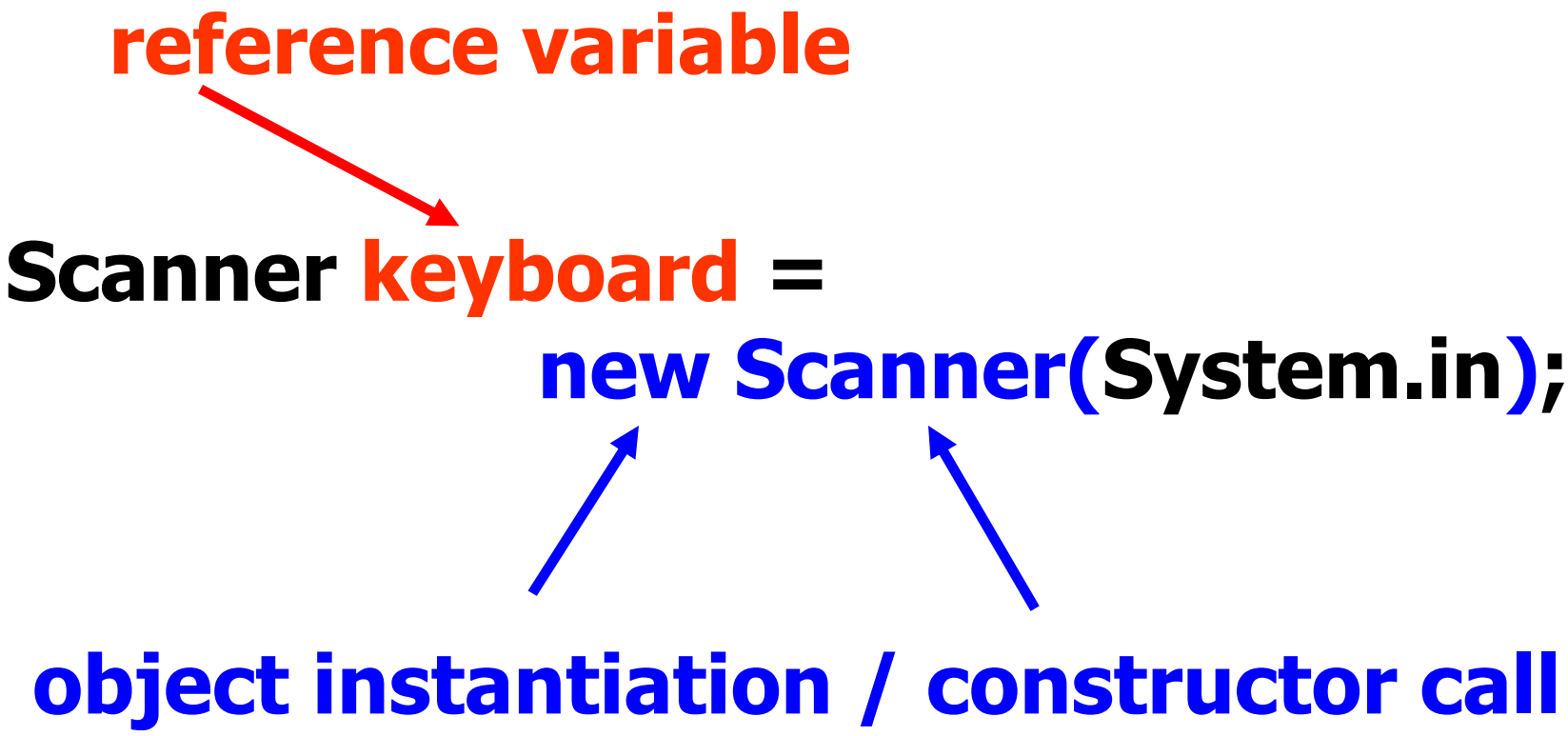
yeller is a reference variable that refers to a Chicken object.

methods




```
public          void        speak(    )
{
  System.out.println("cluck-cluck");
}
```

Constructors



Constructors

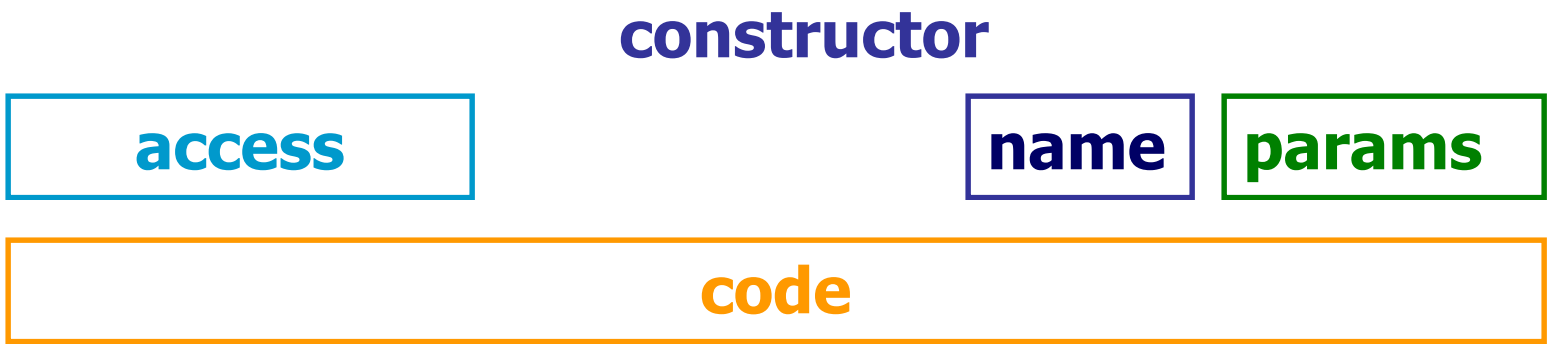
very similar to methods

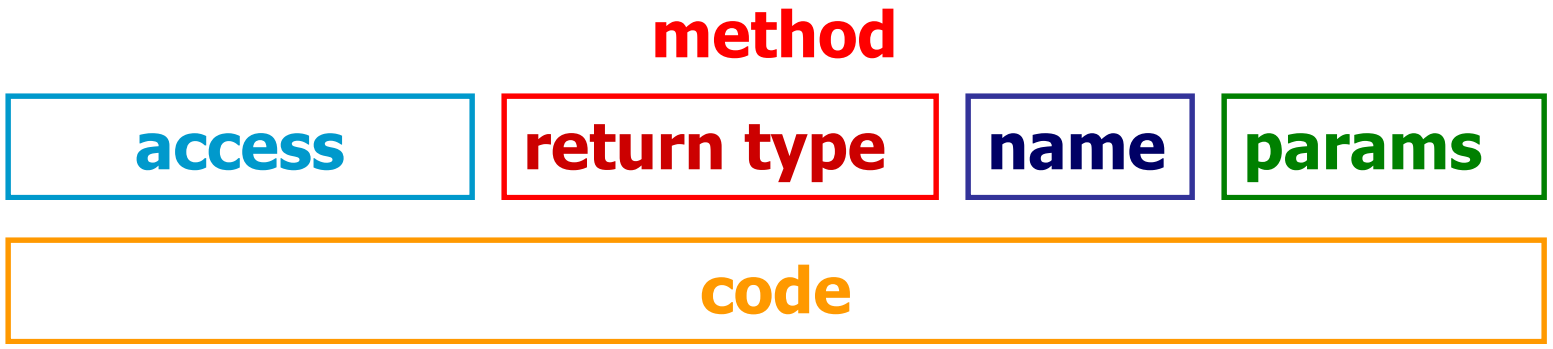
have the same name as the class

have no return type – no void,int,etc.

initialize all instance variables

Constructors VS. Methods





Graphics

frequently used methods

Name	Use
setColor(x)	sets the current drawing color to x
drawString(s,x,y)	draws String s at spot x,y
drawOval(x,y,w,h)	draws an unfilled oval at spot x,y (leftmost point of oval) that is w wide and h tall
fillOval(x,y,w,h)	draws a filled oval at spot x,y (leftmost point of oval) that is w wide and h tall

```
import java.awt.Graphics;
import java.awt.Color;
import javax.swing.JFrame;
```

Graphics

frequently used methods

Name	Use
drawLine(a,b,c,d)	draws a line starting at point a,b and going to point c,d
drawRect(x,y,w,h)	draws an unfilled rectangle at spot x,y (upper left corner) that is w wide and h tall
fillRect(x,y,w,h)	draws a filled rectangle at spot x,y (upper left corner) that is w wide and h tall

```
import java.awt.Graphics;
import java.awt.Color;
import javax.swing.JFrame;
```

Graphics

frequently used methods

Name	Use
drawArc(x,y,w,h,startAngle,arcAngle)	draws an arc at spot x,y that is w wide and h tall
fillArc(x,y,w,h,startAngle,arcAngle)	draws a filled arc at spot x,y that is w wide and h tall
startAngle specifies the start of the arc arcAngle specifies the length of the arc	

```
import java.awt.Graphics;
import java.awt.Color;
import javax.swing.JFrame;
```

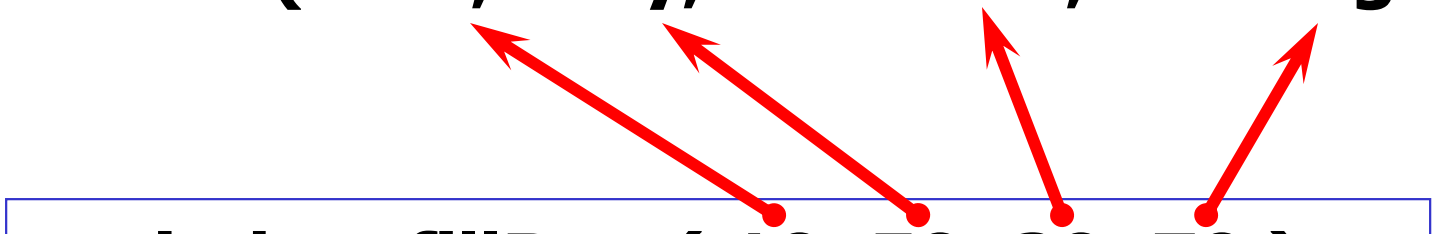
passing parameters

A parameter/argument is a channel used to pass information to a method. setColor() is a method of the Graphics class the receives a Color.

void setColor(Color theColor)

passing parameters

void fillRect (int x, int y, int width, int height)



window.fillRect(10, 50, 30, 70);

method call with parameters

Instance Variables

```
public class InstanceVars
{
  private int one = 8, two = 3;   //instance variables / fields
  private int answer = 0;         //exist throughout the class

  public void add(){
    answer = one + two;
  }

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

  public static void main(String args[])
  {
    InstanceVars test = new InstanceVars();
    test.add();
    test.print();
  }
}
```

OUTPUT

11

What does private mean?

All members with private access can be accessed or modified only inside the class where they are defined.

Encapsulation

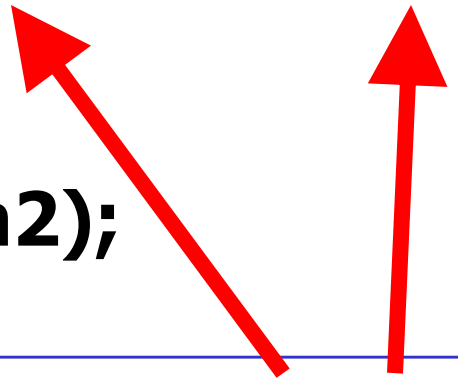
All data members should have private access. A set of public methods should be provided to manipulate the private data.

What does private mean?

All members with private access can be accessed or modified only inside the class where they are defined.

defining parameters

```
public void times(  int num1,   int
num2 )
{
  out.println(num1*num2);
}
```



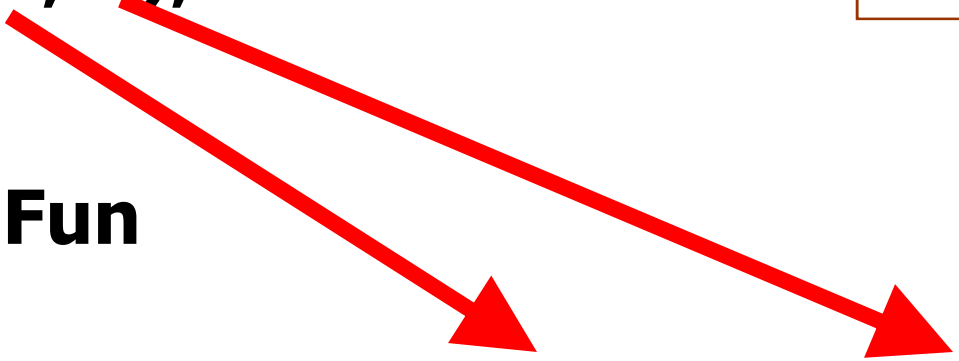
There will be times that we define parameters when we define a method. The parameters allow us to specify the type of data the method will receive.

passing parameters

```
//code in main in another class
Fun test = new Fun();
test.times(3 , 5);
```

OUTPUT

15



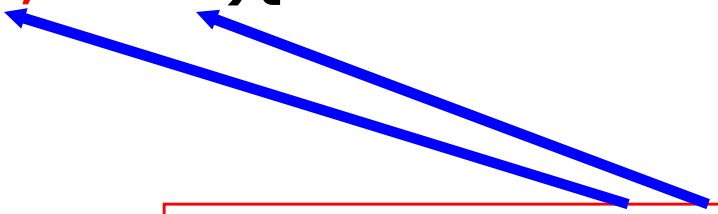
```
public class Fun
{
    public void times(  int num1,   int num2
)
    {
        out.println(num1*num2);
    }
}
```

modifier methods

Modifier methods are methods that change the properties of an object.

modifier methods

```
public class Calc
{
  private int one, two;
  private int answer;

  public void setNums( int n1, int n2 ){
    one=n1;
    two=n2;
  }

  public void add(){
    answer = one + two;
  }

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

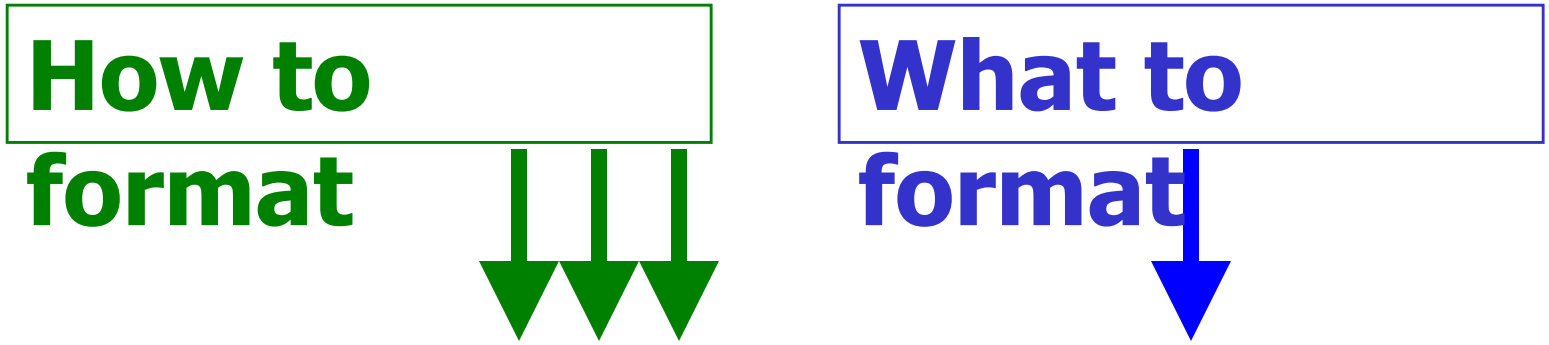


```
test.setNums(4,9
); test.add();
test.print();
```

OUTPUT

13

formatting



OUTPUT

9.24

real format one

```
double dec = 9.231482367;
out.printf("dec == %.1f\n",dec);
out.printf("dec == %.2f\n",dec);
out.printf("dec == %.3f\n",dec);
out.printf("dec == %.4f\n",dec);
out.printf("dec == %.5f\n",dec);
```

OUTPUT

```
dec == 9.2
dec == 9.23
dec == 9.231
dec == 9.2315
dec == 9.23148
```

real format two

```
double dec = 5.3423;
out.println(String.format("%.3f",dec));
out.println(String.format("%12.3f",dec));
out.println(String.format("%-7.3f",dec));
```

OUTPUT

```
5.342
       5.342
5.342  x
```

Operators

+	addition
-	subtraction
*	multiplication
/	division
%	modulus

Integer Math

```
out.println("6 + 5 == " + (6+5));
out.println("6 - 5 == " + (6-5));
out.println("6 * 5 == " + (6*5));
out.println("6 / 5 == " + (6/5));
```

OUTPUT

```
6 + 5 == 11
6 - 5 == 1
6 * 5 == 30
6 / 5 == 1
```

Real Math

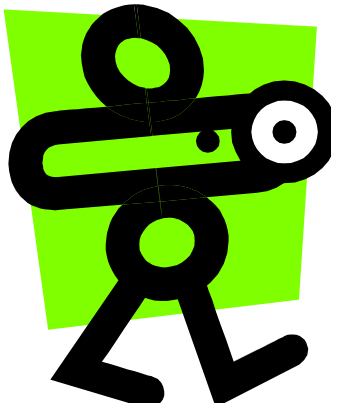
```
out.println("6.1 + 5.2 == " + (6.1+5.2));
out.println("6.1 - 5.2 == " + (6.1-5.2));
out.println("6.1 * 5.2 == " + (6.1*5.2));
out.println("6.1 / 5.2 == " + (6.1/5.2));
```

OUTPUT

```
6.1 + 5.2 == 11.3
6.1 - 5.2 == 0.8999
6.1 * 5.2 == 31.72
6.1 / 5.2 == 1.17307
```

Divide

1/2 = ??
1.0 / 2.0 = ??



1/2 = 0
1 and 2 are integer constants.

1.0/2.0 = 0.5
1.0 and 2.0 are decimal constants.

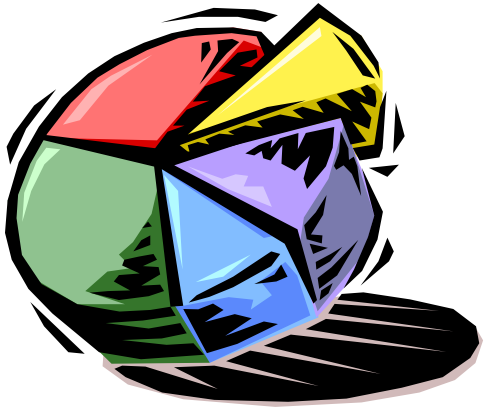
Mod %

mod(%) gives you the integer remainder of integer division.

```
out.println(2 % 3);

out.println(3 % 2);
```

OUTPUT

```
2

1
```



Mod %

mod(%) gives you the real number remainder of real number division.

```
out.println(9 % 3);

out.println(9.2 % 3);
```

OUTPUT

```
0
0.19
```

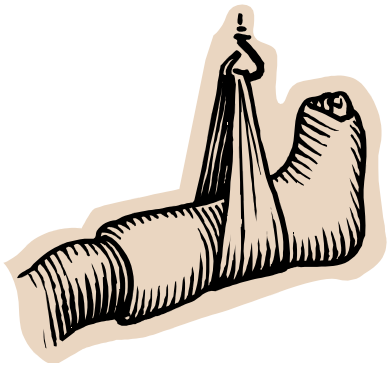
Operator Precedence

()	HIGH
! ++ --	
* / %	
+ -	
= += -= *= /= %=	
,	LOW

Casting

Casting is used to temporarily change the type of a value.

(int)3.14159
(double)3



Casting is often used to create compatibility among data types.

Casting

```
int one = 0;                    //32 bit int
long big = 453;                 //64 bit int
double dec = 7.56;              //64 bit real

one = dec;                      //illegal
one = big;                      //illegal
one = (int)dec;                 //legal
one = (int)big;                 //legal
```

Casting is often used to create compatibility among data types.

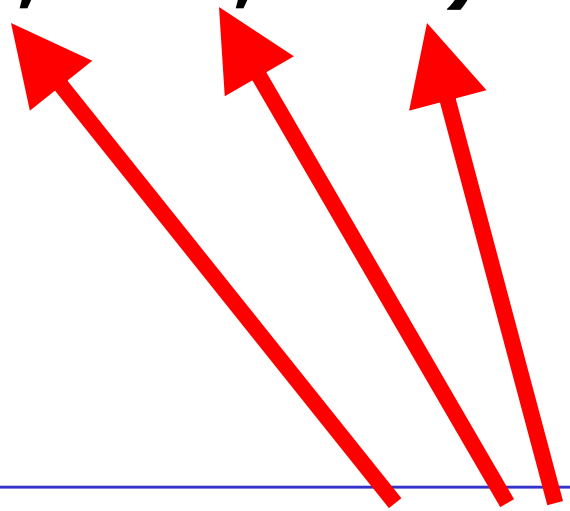
modifier methods

```
public void setSides(int a, int b, int c)
{
    sideA=a;
    sideB=b;
    sideC=c;
}
```

Modifier methods are methods that change the properties of an object.

Initialization Constructor

```
public Triangle(int a, int b, int c)
{
   sideA=a;
   sideB=b;
   sideC=c;
}
```



Constructors often have parameters. The parameters allow data to be passed into the class so that it can be assigned to the instance variables / data fields.

Scope

```
{
   int fun = 99;
}
```

Any variable defined inside of braces, only exists within those braces.

That variable has a scope limited to those braces.

Instance Variables

When you need many methods to have access to the same variable, you make that variable an instance variable.

The scope of an instance variable is the entire class where that variable is defined.

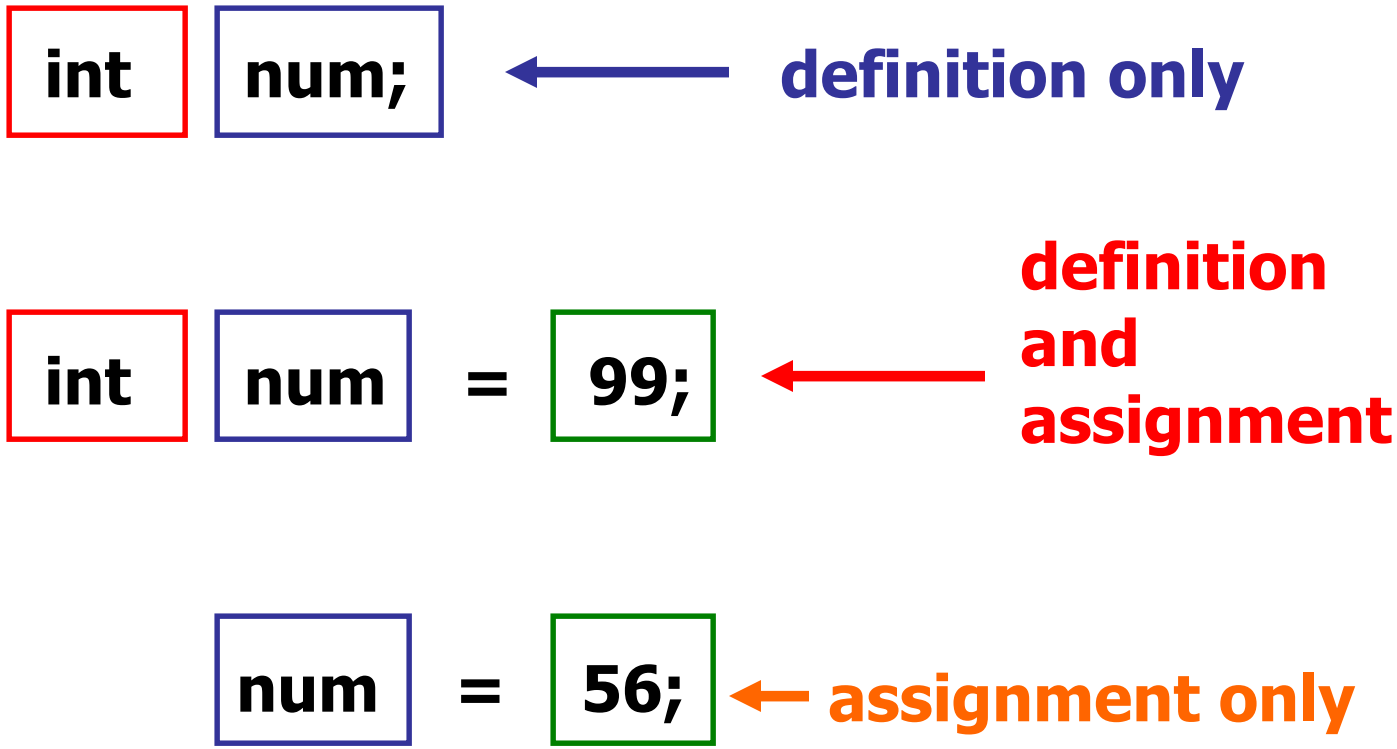
Instance Variables

```
public class InstanceVars
{
  private int one = 8, two = 3;   //instance variables
  private int total = 0;          //exist throughout the class

  public void add(){
    total = one + two;
  }

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

  public static void main(String args[])
  {
    InstanceVars test = new InstanceVars();
    test.add();
    test.print();
  }
}
```

OUTPUT

11

Defining VS. Assigning



Local Variables

When you need only one method to have access to a variable, you should make that variable a local variable.

The scope of a local variable is limited to the method where it is defined.

Local Vars

```
public class LocalVars
{
   private int fun;          //instance variable

   public void change()  {
     int fun = 99;           //local variable
   }

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

   public static void main(String args[])
   {
      LocalVars test = new LocalVars();
      test.change();
      test.print();
   }
}
```

OUTPUT

0

Return Methods

Return methods perform some action and return a result back to the calling location.

```
int num = keyboard.nextInt();
```

nextInt() returns an int back to the calling location.

The value returned is assigned to num.

Return Methods

```
Scanner keyboard =
                new Scanner(System.in);


int num = keyboard.nextInt();
out.println(num);
```

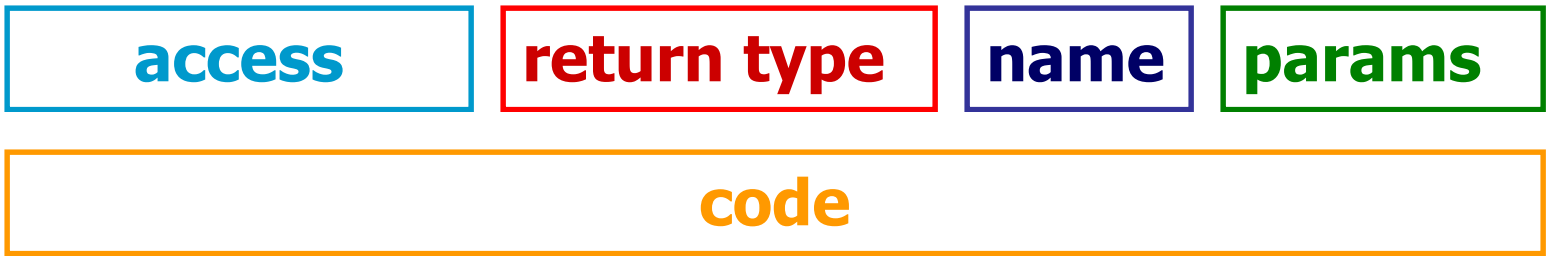
INPUT
1

OUTPUT
1

num
1

return method

Return Method



Math
frequently used methods

Name	Use
floor(x)	rounds x down
ceil(x)	rounds x up
pow(x,y)	returns x to the power of y
abs(x)	returns the absolute value of x
sqrt(x)	returns the square root of x
round(x)	rounds x to the nearest whole number
min(x,y)	returns smallest of x and y
max(x,y)	returns biggest of x and y
random()	returns a double >=0.0 and < 1.0

Math Methods

```
Scanner keyboard =
                new Scanner(System.in);

double num = keyboard.nextDouble();
out.println(Math.ceil(num));
```

INPUT
3.45

OUTPUT
4.0

num
3.45

return methods

Math Methods

```
out.println(Math.floor(3.254));
out.println(Math.ceil(2.45));
out.println(Math.pow(2,7));
out.println(Math.abs(-9));
out.println(Math.sqrt(256));
out.println(Math.sqrt(144));
out.println(Math.round(3.6));
out.println(Math.max(5,7));
out.println(Math.max(5,-7));
out.println(Math.min(5,7));
out.println(Math.min(5,-7));
```

OUTPUT

```
3.0
3.0
128.0
9
16.0
12.0
4
7
5
5
-7
```

constructors

```
public Triangle()
{
   sideA=0;
   sideB=0;
   sideC=0;
}
```

Default Constructor

Constructors are similar to methods. Constructors set the properties of an object to an initial state.

constructors

```
public Triangle(int a, int b, int c)
{
   sideA=a;
   sideB=b;
   sideC=c;
}
```

Initialization Constructor

Constructors are similar to methods. Constructors set the properties of an object to an initial state.

modifier methods

```
public void setSides(int a, int b, int c)
{
    sideA=a;
    sideB=b;
    sideC=c;
}
```

Modifier methods are methods that change the properties of an object.

accessor methods

```
public void print()
{
  out.println(sideA + " " + sideB + " " + sideC);
}
```

Accessor methods are methods that retrieve or grant access to the properties of an object, but do not make any changes.