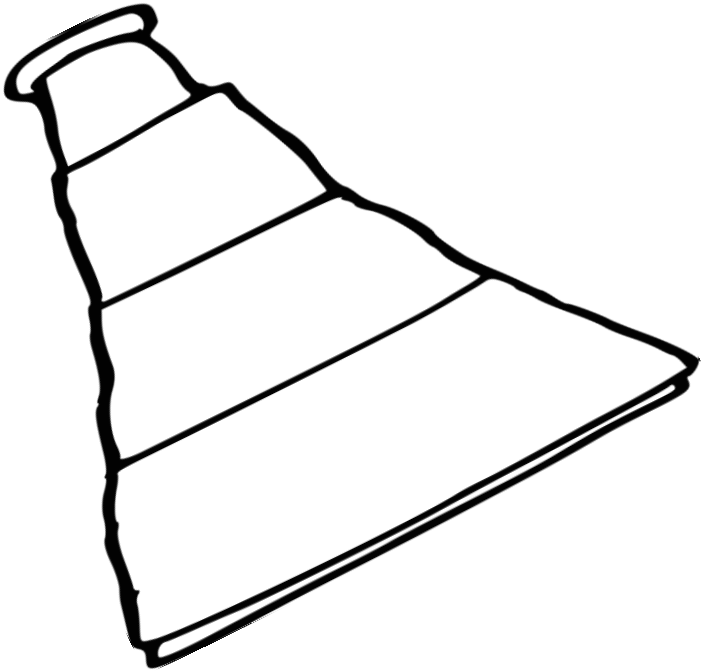


Scope of Variables

Computer Science 112
Boston University

Christine Papadakis

Variable Scope

- The *scope* of a variable is the portion of a program in which the variable can be used.

- By default, the scope of a variable in Java:
  - begins at the point at which it is declared
  - ends at the end of the innermost block that encloses the declaration

```
public static void printResults(int a, int b) {
    System.out.println("Here are the stats:");

    int sum = a + b;
    System.out.print("sum = ");
    System.out.println(sum);

    double avg = (a + b) / 2.0;
    System.out.print("average = ");
    System.out.println(avg);
}
```

Local Variables

- Variables that are declared inside a method are *local variables*.
  - they cannot be used outside that method.

```
public static void printResults(int a, int b) {
    System.out.println("Here are the stats:");

    int sum = a + b;
    System.out.print("sum = ");
    System.out.println(sum);

    double avg = (a + b) / 2.0;
    System.out.print("average = ");
    System.out.println(avg);
}
```

scope of sum

scope of avg

Special Case: Parameters and Variable Scope

- Accessing variables outside of a method?
  - does *not* follow the default scope rules!

```
public class MyClass {
    public static void printResults(int a, int b) {
        System.out.println("Here are the stats:");

        int sum = a + b;
        System.out.print("sum = ");
        System.out.println(sum);

        double avg = (a + b) / 2.0;
        System.out.print("average = ");
        System.out.println(avg);
    }

    int c = a + b;     // does not compile!
}
```

Another Example

```
public class MyProgram {
    public static void method1() {
        int i = 5;
        System.out.println(i * 3);
        int j = 10;
        System.out.println(j / i);
    }

    public static void main(String[] args) {
        // The following line won't compile.
        System.out.println(i + j);

        int i = 4;
        System.out.println(i * 6);
        method1();
    }
}
```

scope of j

scope of method1's version of i

scope of main's version of i

Another Example

1. Variable `i` has not been declared yet.

```
public class MyProgram
    public static void
        int i = 5;
        System.out.prin
        int j = 10;
        System.out.println(j
    }

    public static void main(String[] args) {
        // The following line won't compile.
        System.out.println(i + j);

        int i = 4;
        System.out.println(i * 6);
        method1();
    }
}
```

scope of method1's version of i

scope of main's version of i

Another E

```
public class MyProgram
    public static void
        int i = 5;
        System.out.pri
        int j = 10;
        System.out.println(j
    }

    public static void main(String[] args) {
        // The following line won't compile.
        System.out.println(i + j);

        int i = 4;
        System.out.println(i * 6);
        method1();
    }
}
```

1. Variable `i` has not been declared yet.
2. Variable `j` is not declared in this scope!

pe of
thod1's
version of `i`

scope of `main`'s version of `i`

Another Example

```
public class MyProgram {
    public static void method1() {
        int i = 5;
        System.out.println(i * 3);
        int j = 10;
        System.out.println(j / i);
    }

    public static void main(String[] args) {
        // The following line won't compile.
        // System.out.println(i + j);

        int i = 4;
        System.out.println(i * 6);
        method1();

        System.out.println(i);   // prints 4
    }
}
```

Another Example:

static variables

Memory cell

svar | 10

```
public class MyProgram {
    static int svar = 10;      // declared at class level

    public static void method1() {

        System.out.println(svar);
        svar *= -1;
        System.out.println(svar);
    }

    public static void main(String[] args) {
        System.out.println(svar);
        svar *= 10;
        method1();

        System.out.println(svar);   // prints ?
    }
}
```

Another Example:

static variables

Memory cell

svar | -100

```
public class MyProgram {
    static int svar = 10;

    public static void method1() {

        System.out.println(svar);
        svar *= -1;
        System.out.println(svar);
    }

    public static void main(String[] args) {
        System.out.println(svar);
        svar *= 10;
        method1();

        System.out.println(svar);
    }
}
```

Output

```
10
100
-100
-100
```

Another Example:

static vs. local variable

Memory cell

svar | 10

```
public class MyProgram {
    static int svar = 10; // static variable declaration

    // Declare local paramater svar
    public static void method1( int svar ) {

        System.out.println(svar);
        svar *= -1;
        System.out.println(svar);
    }

    public static void main(String[] args) {
        System.out.println(svar);

        svar *= 10;
        method1( svar );

        System.out.println(svar);
    }
}
```

Another Example:

static vs. local variable

Memory cell

svar | 10

```
public class MyProgram {
    static int svar = 10; // static variable declaration

    // Declare local paramater svar
    public static void method1( int svar ) {

        System.out.println(svar);
        svar *= -1;
        System.out.println(svar)
    }

    pub

}
```

Parameter `svar` is a local variable in `method1` and *hides* the static variable `svar` within the scope of this method.

Another Example:

static vs. local variable

Memory cell

svar | 100

```
public class MyProgram {
    static int svar = 10; // static variable declaration

    // Declare local paramater svar
    public static void method1( int svar ) {

        System.out.println(svar);
        svar *= -1;
        System.out.println(svar);
    }

    public static void main(s
        System.out.println(s

        svar *= 10;
        method1( svar );

        System.out.println(
    }
}
```

Will pass the value of static variable `svar` to `method1`!

Another Example:

static vs. local variable

Memory cell

svar | 100

```
public class MyProgram {
    static int svar = 10; // static variable declaration

    // Declare local paramater svar
    public static void method1( int svar ) {

        System.out.println(svar);
        svar *= -1;
        System.out.println(svar);
    }

    public static void main(String[] args) {
        System.out.println(svar);

        svar *= 10;
        method1( svar );

        System.out.println(svar);
    }
}
```

Output

```
10
100
-100
100
```

Another Version of ChangeAdder!

Another Version of ChangeAdder!

Will It Compile?

```
public class ChangeAdder2 {
    public static void main(String[] args) {
        ...   // earlier code as before
        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;

        if (cents % 100 == 0) {
            int dollars = cents / 100;
            System.out.println(dollars + " dollars");
        } else {
            dollars = cents / 100;
            cents -= dollars*100;
            System.out.println(dollars + " dollars and "
               + cents + " cents");
        }
    }
}
```

How Can We Fix This?

```
public class ChangeAdder2 {
    public static void main(String[] args) {
        ...   // earlier code as before
        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;

        if (cents % 100 == 0) {
            int dollars = cents / 100;
            System.out.println(dollars + " dollars");
        } else {
            dollars = cents / 100;
            cents -= dollars*100;
            System.out.println(dollars + " dollars and "
               + cents + " cents");
        }
    }
}
```

scope of dollars

One Possible Fix: Two Different Variables

```
public class ChangeAdder2 {
    public static void main
        ...   // earlier co
        int cents;
        cents = 25*quarters                                    s;

        if (cents % 100 == 0) {
            int dollars = cents / 100;
            System.out.println(dollars + " dollars");
        } else {
            int dollars = cents / 100;
            cents -= dollars*100;
            System.out.println(dollars + " dollars and "
               + cents + " cents");
        }
    }
}
```

Will this code compile now?

scope of 1st dollars

scope of 2nd dollars

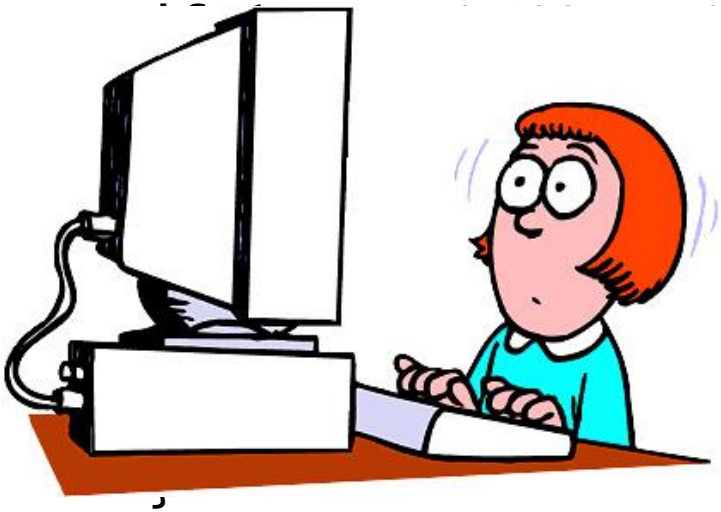
One Possible Fix: Two Different Variables

```
public class ChangeAdder2 {
    public static void main
        ...   // earlier co
        int cents;
        cents = 25*quarters                                    s;

                                                  ) {
                                                  ents / 100;
                                                  :ln(dollars + " dollars");

                                                  ents / 100;
                                                   *100;
                                                  :ln(dollars + " dollars and "
                                                  ents");
    }
}
```

scope of 1st dollars

scope of 2nd dollars

Yes, but anyone reading this code would be doing this! Why duplicate the declarations?



One Possible Fix: Two Different Variables

```
public class ChangeAdder2 {
    public static void main(String[] args) {
        ...   // earlier code as before
        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;

        if (cents % 100 == 0) {
            int dollars = cents / 100;
            System.out.println(dollars + " dollars");
        } else {
            int dollars = cents / 100;
            cents = dollars *100;
            System.out.println(dollars + " dollars and "
               + cents + " cents");
        }
    }
}
```

scope of 1st dollars

scope of 2nd dollars

And if your program needed access to the variable outside of the if..else block it would not have it.

Another Possible Fix: Move the Declaration

```
public class ChangeAdder2 {
    public static void main(String[] args) {
        ...   // earlier code as before
        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;

        int dollars;
        if (cents % 100 == 0) {
            dollars = cents / 100;
            System.out.println(dollars + " dollars");
        } else {
            dollars = cents / 100;
            cents -= dollars*100;
            System.out.println(dollars + " dollars and "
               + cents + " cents");
        }
    }
}
```

- The scope for variable `dollars`.

Another Recommended Change

```
public class ChangeAdder2 {
    public static void main
        ...   // earlier co
        int cents;
        cents = 25*quarters                                    s;

        int dollars;
        if (cents % 100 == 0) {
            dollars = cents / 100;
            System.out.println(dollars + " dollars");
        } else {
            dollars = cents / 100;
            cents -= dollars*100;
            System.out.println(dollars + " dollars and "
               + cents + " cents");
        }
    }
}
```

Why repeat this statement in each block?

Another Recommended Change

```
public class ChangeAdder2 {
    public static void main
        ...   // earlier co
        int cents;
        cents = 25*quarters                                    s;

        int dollars = cents / 100;
        if (cents % 100 == 0) {
            System.out.println(dollars + " dollars");
        } else {
            cents -= dollars*100;
            System.out.println(dollars + " dollars and "
               + cents + " cents");
        }
    }
}
```

Why not move it up and perform the assignment once here as well?

Another Recommended Change

```
public class ChangeAdder2 {
    public static void main
        ...   // earlier co
        int cents;
        cents = 25*quarters                                    s;

        int dollars = cents / 100;
        if (cents % 100 == 0) {
            System.out.println(dollars + " dollars");
        } else {
            cents -= dollars*100;
            System.out.println(dollars + " dollars and "
               + cents + " cents");
        }
    }
}
```

How about these statements?

Special Case: `for` Loops and Variable Scope

- When a variable is declared in the initialization clause of a `for` loop, its scope is limited to the loop.

- Example:

```
public static void myMethod() {
    for (int i = 0; i < 5; i++) {
        int j = i * 3;
        System.out.println(j);
    }
    // the following line won't compile
    System.out.print(i);
    System.out.println(" values were printed.");
}
```

scope of `i`

Special Case: `for` Loops and Variable Scope

- To allow `i` to be used outside the loop, we need to declare it outside the loop:

- Example:

```
public static void myMethod() {
    int i;
    for (i = 0; i < 5; i++) {
        int j = i * 3;
        System.out.println(j);
    }

    // now this will compile
    System.out.print(i);
    System.out.println(" values were printed.");
}
```

scope of `i`

Special Case: `for` Loops and Variable Scope

- To allow `i` to be used outsi
  declare it outside the loo

This is fine as long as we need to access the variable outside the for loop!

- Example:

```
public static void myM  hod
    int i;
    for (i = 0; i < 5; i++) {
        int j = i * 3;
        System.out.println(j);
    }

    // now this will compile
    System.out.print(i);
    System.out.println(" values were printed.");
}
```

scope of `i`

Special Case: `for` Loops and Variable Scope

- To allow `i` to be use
  declare it outside

Note, this is an error in the current version of Java!

- Example:

```
public static void yMethod() {
    int i;
    for (int i = 0; i < 5; i++) {
        int j = i * 3;
        System.out.println(j);
    }

    System.out.print(i);
    System.out.println(" values were printed.");
}
```

Special Case: `for` Loops and Variable Scope (cont.)

- Limiting the scope of the control variable is more efficient and allows us to use the same variable multiple times in the same method.

- Example:

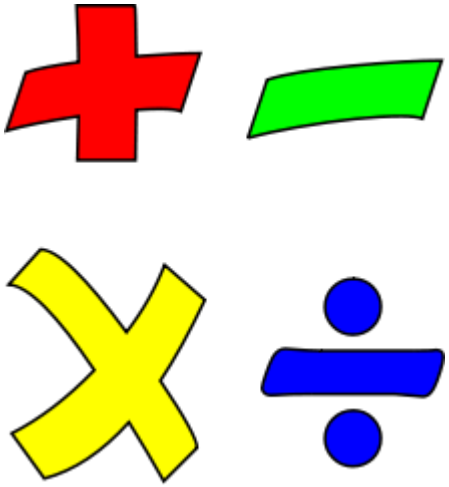
```
public static void myMethod() {
    for (int i = 0; i < 5; i++) {
        int j = i * 3;
        System.out.println(j);
    }

    for (int i = 0; i < 7; i++) {
        System.out.println("Go BU!");
    }
}
```

scope of first `i`

scope of second `i`

Practice with Scope

```
public static void drawRectangle(int height) {
    for (int i = 0; i < height; i++) {
        // which variables could be used here?   height, i
        int width = height * 2;
        for (int j = 0; j < width; j++) {
            System.out.print("*");
            // what about here?   height, i, width, j
        }
        // what about here?   height, i, width
        System.out.println();
    }
    // what about here?   height
}

public static void repeatMessage(int numTimes) {
    // what about here?   numTimes
    for (int i = 0; i < numTimes; i++) {
        System.out.println("What is your scope?");
    }
}
```



From Python to Java: Operations on Data Types

Computer Science 112
Boston University

Christine Papadakis

Recall: Our *original* Change-Adder Program

```
import java.util.*;

public class ChangeAdder {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Enter the number of quarters: ");
        int quarters = console.nextInt();
        System.out.print("Enter the number of dimes: ");
        int dimes = console.nextInt();
        System.out.print("Enter the number of nickels: ");
        int nickels = console.nextInt();
        System.out.print("Enter the number of pennies: ");
        int pennies = console.nextInt();

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
    }
}
```

What computational statement can we add to compute the ***total*** **dollar amount**?

An Incorrect Extended Version

```
import java.util.*;

public class ChangeAdd
    public static void
        Scanner consol

        System.out.pri
        int quarters =
        System.out.pri
        int dimes = co
        System.out.pri
        int nickels =
        System.out.pri
        int pennies =

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        int dollars = cents / 100;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

Does this statement correctly calculate the amount we are interested in?

An Incorrect Extended Version

```
import java.util.*;

public class ChangeAdd
    public static void
        Scanner consol

        System.out.pri
        int quarters =
        System.out.pri
        int dimes = co
        System.out.pri
        int nickels =
        System.out.pri
        int pennies =

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        int dollars = cents / 100;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

No the integer data type cannot store a floating point value!

An Incorrect Extended Version

```
import java.util.*;

public class ChangeAdd
    public static void
        Scanner consol

        System.out.pri
        int quarters =
        System.out.pri
        int dimes = co
        System.out.pri
        int nickels =
        System.out.pri
        int pennies =

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        double dollars = cents / 100;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

Does this statement correctly calculate the amount we are interested in?

An Incorrect Extended Version

```
import java.util.*;

public class ChangeAdd
    public static void
        Scanner consol

        System.out.pri
        int quarters =
        System.out.pri
        int dimes = co
        System.out.pri
        int nickels =
        System.out.pri
        int pennies =

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        double dollars = cents / 100;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

Still no! The decimal precision is lost even though we have declared the variable *dollars* to be a double.

An Incorrect Extended Version

```
import java.util.*;

public class ChangeAdder2 {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

                                    mber of quarters: ");
                                    ();
                                    mber of dimes: ");

                                    mber of nickels: ");
                                    );
                                    mber of pennies: ");
                                    );

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        double dollars = cents / 100;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

In Java the operation that is applied is dependent on the operands.

An Incorrect Extended Version

```
import java.util.*;

public class ChangeAdder2 {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

                                    mber of quarters: ");
                                    ();
                                    mber of dimes: ");

                                    mber of nickels: ");
                                    );
                                    mber of pennies: ");
                                    );

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        double dollars = cents / 100;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

In this case, as both our operands are integers, integer division is performed.

An Incorrect Extended Version

```
import java.util.*;

public class ChangeAdder2 {
    rgs) {
    System.in);

    mber of quarters: ");
    ();
    mber of dimes: ");

    mber of nickels: ");
    );
    mber of pennies: ");
    );

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        double dollars = cents / 100;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

In order for floating point division to be performed, one of the operands must be a double (either as a declared variable or as a literal value).

A *Corrected* Extended Version

```
import java.util.*;

public class ChangeAdder2 {
                                                rgs) {
                                                System.in);

                                                mber of quarters: ");
                                                ();
                                                mber of dimes: ");

                                                mber of nickels: ");
                                                );
                                                mber of pennies: ");
                                                );

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        double dollars = cents / 100.0;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

Our options are:

1. Change the literal value of 100 to 100.0

A *Corrected* Extended Version

```
import java.util.*;

public class ChangeAdder2 {
                                                rgs) {
                                                System.in);

                                                mber of quarters: ");
                                                ();
                                                mber of dimes: ");

                                                mber of nickels: ");
                                                );
                                                mber of pennies: ");
                                                );

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        double dollars = (double) cents / 100;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

Our options are to:

1. change the literal value of 100 to 100.0 or
2. **cast** variable cents to a double!

A *Corrected* Extended Version

*Note casting a variable **does not** change the physical data type of the variable, it just allows the system to operate on it (within the context of a specific statement) as if it were of the specified type.*

```
im

pu                                  gs) {
                                    ystem.in);

                                    per of quarters: ");
                                    );
                                    per of dimes: ");

                                    per of nickels: ");
                                    ;
                                    per of pennies: ");
        int pennies = console.nextint();

        int cents;
        cents = 25*quarters + 10*dimes + 5*nickels + pennies;
        System.out.println("total in cents is: " + cents);
        double dollars = (double) cents / 100;
        System.out.print("total in dollars is: $" + dollars);
    }
}
```

Another Incorrect Extended Version

```
import java.util.*;

public class ChangeAdder3 {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
```

What would happen if we wrote the following statement in Java:

int dollars = (double) cents/100 ?

1. Would the correct amount be computed? **Yes**

2. Would the correct amount be stored? **NO! dollars is declared as an integer, therefore the precision is truncated.**

Operators and Data Types

- Each data type has its own set of operators.
  - the `int` version of an operator produces an `int` result
  - the `double` version produces a `double` result
  - etc.

- Rules for numeric operators:
  - if the operands are both of type `int`,
    the `int` version of the operator is used.
    - examples:
      ```
      15 + 30
      1 / 2
      25 * quarters  // quarters is an int
      ```
  - if at least one of the operands is of type `double`,
    the `double` version of the operator is used.
    - examples:
      ```
      15.5 + 30.1
      1 / 2.0
      25.0 * quarters
      ```

Two Types of Division

Example

- The `int` version of `/` performs *integer division*, which discards everything after the decimal.
  - like `//` in Python

- The `double` version of `/` performs *floating-point division*, which keeps the digits after the decimal.

- Examples:

statement	output
`System.out.println(5 / 3.0);`	`1.6666666666666667`
`System.out.println(5 / 3);`	`1`
`System.out.println(16.0 / 5);`	`3.2`
`System.out.println(16 / 5);`	`3`

Java Values and Types

type	set of values	literal values	operations
`char`	characters	`'A'` `'@'`	compare
`String`	sequences of characters	`"Hello World"` `"126 is fun"`	concatenate
`int`	integers	`17` `12345`	add, subtract, multiply, divide
`double`	floating-point numbers	`3.1415` `6.022e23`	add, subtract, multiply, divide
`boolean`	truth values	`true` `false`	and, or, not

Overloaded Operators

Some Java Operators are **overloaded** to perform the appropriate operation based on the data type.

Example: +

Java Operators

- When Java evaluates an overloaded operator, it automatically performs *widening* conversions as necessary to *make the operands fit the operator*:

Example: + is overloaded – it works for two ints or two doubles….

```
public static void main(String[] args) {

    int x = 4;

    double y = 2.3;

    System.out.println( ( x + x ) );     // prints:   8

    System.out.println( ( y + y ) );     // prints:   4.6

    System.out.println( ( x + y ) );     // prints:   6.3

}
```

All the arithmetic operators in java are overloaded for int and double.

Java Operators

- When Java evaluates an overloaded operator, it automatically performs *widening* conversions as necessary to *make the operands fit the operator*:

Example: + is overloaded – it works for two ints or two doubles….

```
public static void main(String[] args) {

    int x = 4;

    double y = 2.3;

    System.out.println( ( x + x ) );     // prints:   8

    System.out.println( ( y + y ) );     // prints:   4.6

    System.out.println( ( x + y ) );     // prints:   6.3

}
```

All the arithmetic operators in java are overloaded for int and double.

Widening Conversion:

4 + 2.3

4.0 + 2.3 => 6.3

Result is the wider type!

Java Operators

Division is overloaded, therefore behaves differently for ints and doubles…..

Java: division operator is “overloaded”:

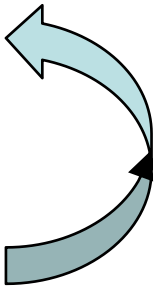
/ returns an int if both operands are ints, otherwise returns double:

5 / 2 => 2

5.0 / 2.0 => 2.5

5.0 / 2 => 2.5

5 / (double) 2 => 2.5



In both cases, the 2 is widened to 2.0!

Java Values and Types