# Quick Check

In what order are the operators evaluated in the following expressions?

```
a + b + c + d + e
```

```
a + b * c - d / e
```

```
a / (b + c) - d % e
```

```
a / (b * (c + (d - e)))
```

# Quick Check

In what order are the operators evaluated in the following expressions?

```
a + b + c + d + e
  1   2   3   4
```

```
a + b * c - d / e
  3   1   4   2
```

```
a / (b + c) - d % e
  2    1     4   3
```

```
a / (b * (c + (d - e)))
  4    3    2    1
```

# Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
```

4 1 3 2

Then the result is stored in the variable on the left hand side

# Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of `count`

```
count  =  count + 1;
```

Then the result is stored back into `count` (overwriting the original value)

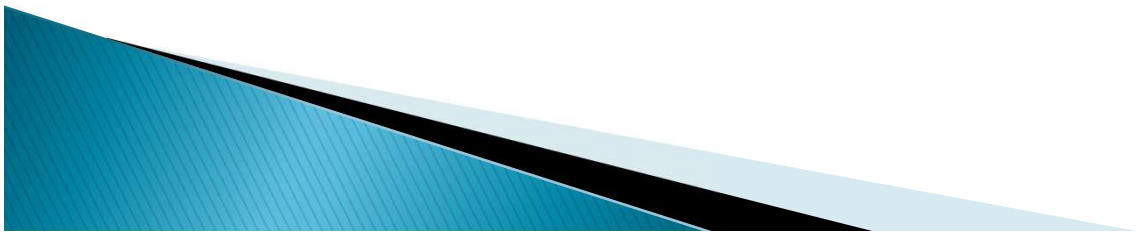
# Increment and Decrement

- The increment (++) and decrement (--) operators use only one operand
- The statement

```
count++;
```

  is functionally equivalent to

```
count = count + 1;
```



# Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

# Assignment Operators

- There are many assignment operators in Java, including the following:

| Operator | Example | Equivalent To |
|---|---|---|
| `+=` | `x += y` | `x = x + y` |
| `-=` | `x -= y` | `x = x - y` |
| `*=` | `x *= y` | `x = x * y` |
| `/=` | `x /= y` | `x = x / y` |
| `%=` | `x %= y` | `x = x % y` |

# Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore

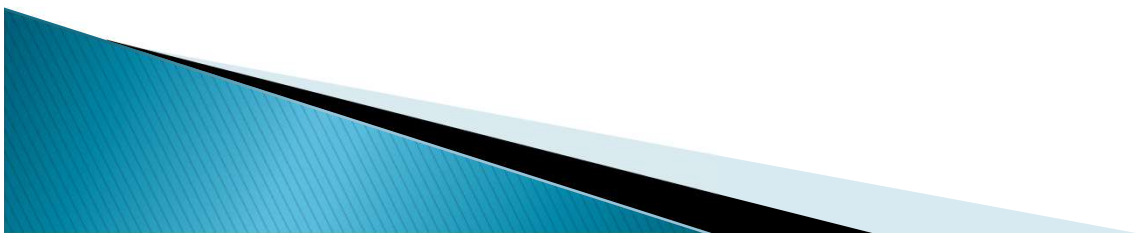
```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```

# Assignment Operators

- The behavior of some assignment operators depends on the types of the operands
- If the operands to the += operator are strings, the assignment operator performs string concatenation
- The behavior of an assignment operator (+=) is always consistent with the behavior of the corresponding operator (+)



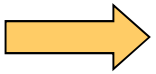
# Outline

**Variables and Assignment**

**Primitive Data Types**

**Character Strings**

**Expressions**

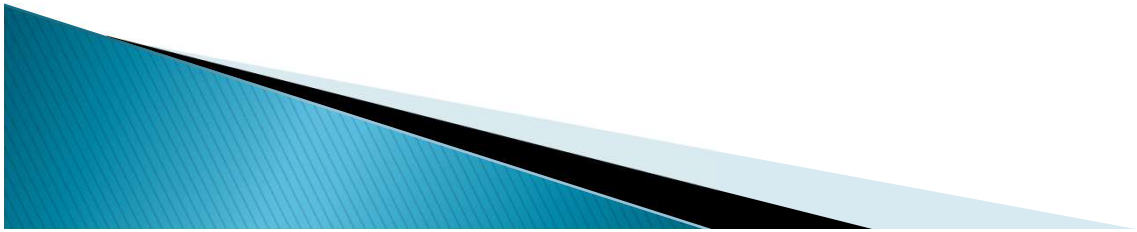


**Data Conversion**

**Interactive Programs**

# Data Conversion

- Sometimes it is convenient to convert data from one type to another
- For example, in a particular situation we may want to treat an integer as a floating point value
- These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation



# Data Conversion

- *Widening conversions* are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)

- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)
- In Java, data conversions can occur in three ways:
  - assignment conversion
  - promotion
  - casting

# Data Conversion

Widening Conversions

| From | To |
|---|---|
| byte | short, int, long, float, or double |
| short | int, long, float, or double |
| char | int, long, float, or double |
| int | long, float, or double |
| long | float or double |
| float | double |

Narrowing Conversions

| From | To |
|---|---|
| byte | char |
| short | byte or char |
| char | byte or short |
| int | byte, short, or char |
| long | byte, short, char, or int |
| float | byte, short, char, int, or long |
| double | byte, short, char, int, long, or float |

# Assignment Conversion

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
- Example:

```
int dollars = 20;
double money = dollars;
```

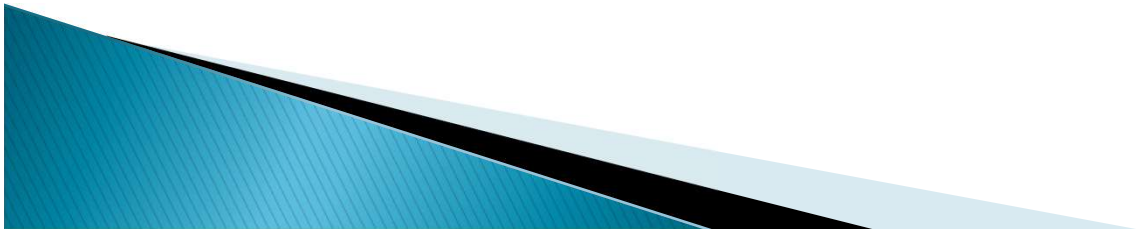
- Only widening conversions can happen via assignment
- Note that the value or type of `dollars` did not change

# Promotion

- *Promotion* happens automatically when operators in expressions convert their operands
- Example:

```
int count = 12;
double sum = 490.27;
result = sum / count;
```

- The value of `count` is converted to a floating point value to perform the division calculation



# Casting

- *Casting* is the most powerful, and dangerous, technique for conversion
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted

```
int total = 50;
float result = (float) total / 6;
```

- Without the cast, the fractional part of the answer would be lost

# Quick check

- What is the result of the following assignments?

```
int a = 7, b = 3;
double c, d;
c = a / b;
d = (double) a / b;
```

# Quick check

- What is the result of the following assignments?

```
int a = 7, b = 3;
double c, d;
c = a / b;
d = (double) a / b;
```

**c = 2.0**

**d = 2.3333333333333333**

# Outline

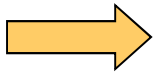
**Variables and Assignment**

**Primitive Data Types**

**Character Strings**

**Expressions**

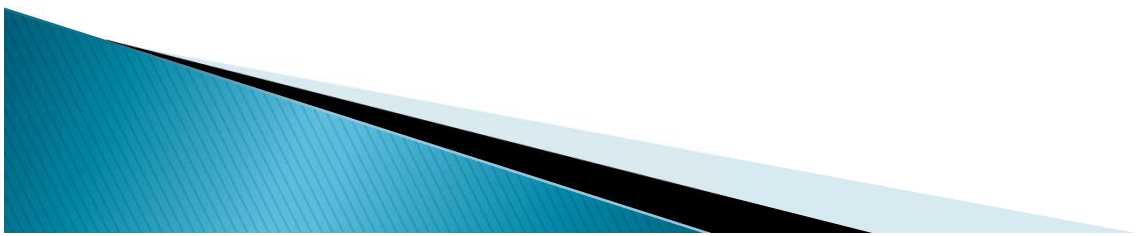
**Data Conversion**



**Interactive Programs**

# Interactive Programs

- Programs generally need input on which to operate
- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object



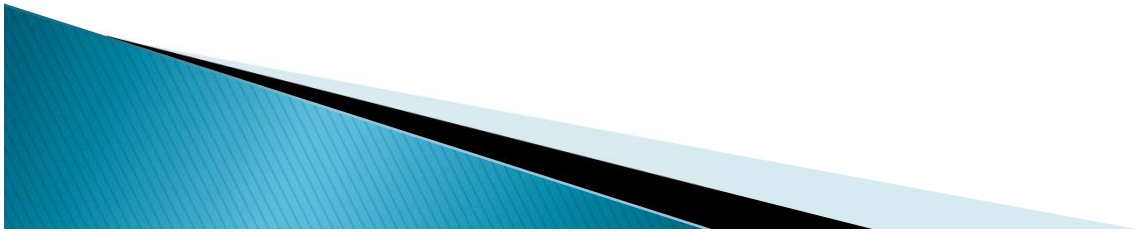
# Reading Input

- The following line creates a `Scanner` object that reads from the keyboard:

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

- The `new` operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
answer = scan.nextLine();
```



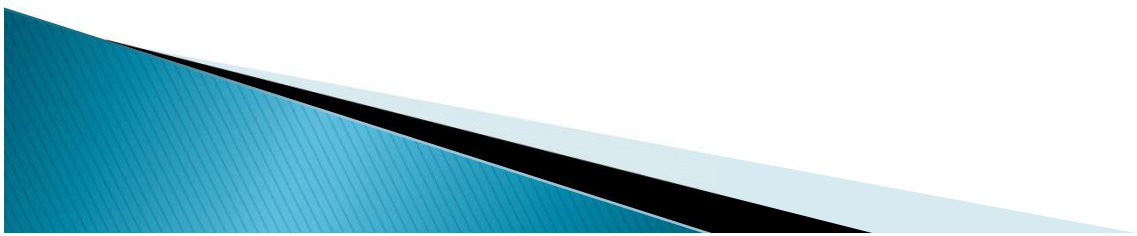
# Reading Input

- The `Scanner` class is part of the `java.util` package, and must be imported in order to be used
- All import statements must be inserted in the class before the class header:

```
import java.util.Scanner;
public class Example {
…
}
```

- You can use the wildcard * to import all the classes within a package:

```
import java.util.*;
```



# Some methods from Scanner class

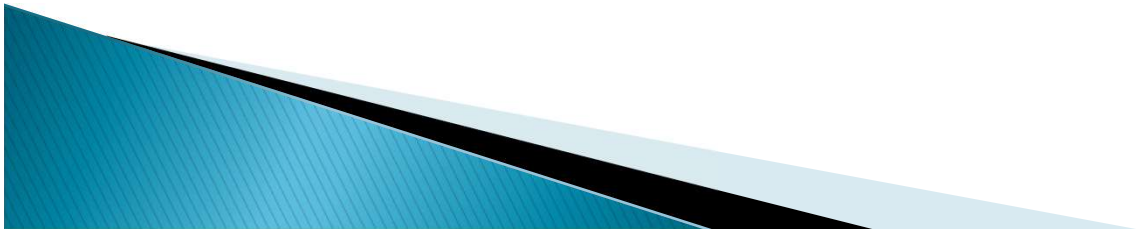
| Data Type | Corresponding method |
|---|---|
| byte | nextByte() |
| short | nextShort() |
| int | nextInt() |
| long | nextLong() |
| float | nextFloat() |
| double | nextDouble() |
| boolean | nextBoolean() |
| char | next().charAt(0) |
| String | next()<br>nextLine() |

# Some methods from Scanner class

Scanner class uses whitespace (spaces, tabs, new-lines) to extract a token from the source. We call the whitespace (shown as \s) as the default delimiter. The delimiter can be changed through the **useDelimiter**

Method. Example:

```
Scanner scan = new Scanner
(System.in);
scan.useDelimiter("; ");
```



# Some methods from Scanner class

- Normally, the Scanner class uses the computers own Locale settings to convert a token into its data type such as a double. But, US Locale uses a . as a decimal point whereas Turkish Locale uses a , as a decimal point. If you use the wrong decimal point, you may end up with a run-time error when using nextFloat() or nextDouble()
- Use `scan.useLocale(Locale.US)` to avoid such problems and hence always use . as the decimal point! But, you must also import the Locale class:

```
import java.util.Locale;
```

# nextLine() method of Scanner class

The nextLine method reads all of the input until the end of the line is found.
See Echo.java
You must be careful when using nextLine() after using any of the other next(), nextInt(), nextDouble(),.. methods. You may need to call the nextLine() method twice! (First one consumes the new line character not consumed before by the non-nextLine() method, and the second one reads the real string until end-of-line character.)
Alternatively, one may prefer to reserve one Scanner object for nextLine() and another for non-nextLine() methods!

```
//********************************************************************
//  Echo.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the nextLine method of the Scanner class
//  to read a string from the user.
//********************************************************************

import java.util.Scanner;

public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String message;
      Scanner scan = new Scanner (System.in);

      System.out.println ("Enter a line of text:");

      message = scan.nextLine();

      System.out.println ("You entered: \"" + message + "\"");
   }
}
```

## Sample Run

```
Enter a line of text:
You want fries with that?
You entered: "You want fries with that?"
```

```
//********************************************************************
//  Ec                                                                ***
//
//  De                                                                s
//  to
//********************************************************************

import java.util.Scanner;

public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String message;
      Scanner scan = new Scanner (System.in);

      System.out.println ("Enter a line of text:");

      message = scan.nextLine();

      System.out.println ("You entered: \"" + message + "\"");
   }
}
```

# Input Tokens

- Unless specified otherwise (by the useDelimiter method) , *white space* is used to separate the elements (called *tokens*) of the input
- White space includes space characters, tabs, new line characters
- The `next` method of the `Scanner` class reads the next input token and returns it as a string
- Methods such as `nextInt` and `nextDouble` read data of particular types
- See `GasMileage.java`

```
//********************************************************************
//  GasMileage.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the Scanner class to read numeric data.
//********************************************************************

import java.util.Scanner;

public class GasMileage
{
   //-----------------------------------------------------------------
   //  Calculates fuel efficiency based on values entered by the
   //  user.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int miles;
      double gallons, mpg;

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

continue

continue

```
        System.out.print ("Enter the number of miles: ");
        miles = scan.nextInt();

        System.out.print ("Enter the gallons of fuel used: ");
        gallons = scan.nextDouble();

        mpg = miles / gallons;

        System.out.println ("Miles Per Gallon: " + mpg);
    }
}
```

continue

```
        Sy
        mi

        Sy
        gallons = scan.nextDouble();

        mpg = miles / gallons;

        System.out.println ("Miles Per Gallon: " + mpg);
    }
}
```

## Sample Run

```
Enter the number of miles: 328
Enter the gallons of fuel used: 11.2
Miles Per Gallon: 29.28571428571429
```

# More on Scanner

- See UserAddition.java

# Summary

- character strings
- primitive data
- the declaration and use of variables
- expressions and operator precedence
- data conversions
- accepting input from the user