

Mat 2170 Chapter Three

Java Expressions

Spring 2012

Student Responsibilities

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

- Reading: Textbook, Chapters 3
- Lab 2, due Thursday in Lab 3
 - 1 Electronic Submission (`submit`)
 - 2 Publish to Web (`websync`)
 - 3 Postlab and printouts
- Lab 3
 - 1 Prelab - due at beginning of Lab 3
 - 2 Postlab and printouts - due in Lab 4
- Attendance

Chapter Three — Expressions

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment
Applications

Boolean
Expressions

Data Types

- 3.1 Primitive data types
- 3.2 Constants and variables
- 3.3 Operators and operands
- 3.4 Assignment statements
- 3.5 Boolean expressions
- 3.6 Designing for change

Expressions in Java

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

Consider the Java statement: `int total = n1 + n2;`

- This statement combines the values in the integer objects `n1` and `n2` and places the sum in the integer object `total`
- The `n1 + n2` that appears on the right hand side (RHS) of the **assignment operator (=)** is an example of an expression, which specifies the operations involved in the computation.
- A Java expression consists of terms joined together by operators.

Valid Java Expression Terms

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

- A **constant** such as `3.141592` or `"Hello, World!"`
- An **object name** such as `n1`, `n2`, or `total`
- A **method call** that returns a value, such as `readInt()`
- An **expression enclosed in parentheses**

Constants and Variables

- The simplest terms that appear in expressions are **constants** and **variables**.
- The value of a **constant does not change** during the course of a program. Go figure!
- Declaring **constants** (by convention, between the last curly braces of the `run()` method and the class):

```
private static final int MULT1 = 3;  
private static final double PI = 3.14159;  
private static final double TWO_PI = 2 * PI;
```

- **DO NOT USE “MAGIC NUMBERS”**
(aka un-named constants).

Variables

- A **variable** is the name used to identify the piece of memory set aside during execution to hold a value; it can be updated as the program runs.
- Each variable has three attributes:
 - **name**: differentiates one variable from another
 - **type**: specifies type of value variable can contain
 - **value**: current contents of the variable
- The **name** and **type** of a variable are **fixed**.
- The **value** changes whenever you assign a new value to the variable.

Differences in Storage

Mat 2170
Chapter
Three

Java
Expressions

total: *an int = 42*

42

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

minimum: *a float = 0.001096*

-3	(1).0960000000000000
----	----------------------

↑ ↑ ↑
| | |
sign exponent mantissa

Variables may be visualized as naming various sizes of shoe boxes (depending on type), and are capable of storing a single value.

Java Identifiers

- Names for variables (and other things) are called **identifiers**.
- **Identifiers** in Java conform to the following rules:
 - Must begin with a letter or underscore
 - Remaining characters must be letters, digits, or underscore
 - Must not be one of Java's reserved words
 - Should make their purpose obvious to the reader (programmer)
 - Should adhere to standard conventions. (Variable names, for example, usually begin with a lowercase letter; while constants are all uppercase.)

Java Reserved Words

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

These may not be used as identifiers

abstract	else	interface	super
boolean	extends	long	switch
break	false	native	synchronized
byte	final	new	this
case	finally	null	throw
catch	float	package	throws
char	for	private	transient
class	goto	protected	true
const	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while
double	int	strictfp	

Variable Declarations

- A variable must be **declared** before it can be used.
- The declaration establishes the **type** and **name** of the variable.
- A common form of a variable declaration is:

```
type name;
```

where

- **type** is the name of a primitive type or a class
 - **name** is a unique identifier
 - Note that no value is assigned to the variable — it is considered to be **uninitialized**.
- After a variable is declared, it may then be initialized with an assignment statement:

```
name = value;
```

Variable Declarations *with* Initialization

- The declaration may also establish the **type**, **name**, and **initial value**, all in one statement.

- Another common form of a variable declaration is:

```
type name = value ;
```

where

- **type** is the name of a primitive type or a class
 - **name** is a unique identifier, and
 - **value** is an expression specifying the initial value
- The variable can be given another value in an assignment statement, just as before.

Equivalent Examples

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

```
// Declare, then initialize
int n1;
int n2;
int total;
n1 = readInt("Enter first integer: ");
n2 = readInt("Enter second integer: ");
total = n1 + n2;
```

```
// Declare and initialize in same statement
int n1 = readInt("Enter first integer: ");
int n2 = readInt("Enter second integer: ");
int total = n1 + n2;
```

Variable Declarations in Class Methods

- Most declarations appear as statements in the body of a method definition. Two examples we've seen:

```
int n1 = readInt("Enter first number:  ");
```

```
GRect Face =  
    new GRect(LeftX, TopY,  
              FaceWidth, FaceHeight);
```

- Variables declared in methods are called **local variables** and are accessible (known) only inside the method where they are declared.
- Variables may also be declared as part of a class, and then are called **instance variables** (or **data members** of the class).

Operators and Operands

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

- Java arithmetic expressions closely resemble mathematical expressions.
- The most common operators are the ones that specify arithmetic computation:

+	Addition	*	Multiplication
-	Subtraction	/	Division
		%	Remainder

- Operators usually appear between two sub-expressions, called **operands**
- **binary operator**: an operator that take two operands.
- The $-$ (minus) operator can also appear as a **unary** operator, as in the expression $-x$, which denotes the negative of x .

Integer Division

Whenever you apply a binary operator to numeric values, the result will be of type **int** if **both** operands are of type **int**, but will be **double** if either operand (or both) is **double**.

- **This rule has important consequences** in the case of division — when two integers are the operands, the resulting quotient is **truncated**, meaning **cut off!**

- **Beware of Integer Division:**

```
3 / 5   is 0
5 / 3   is 1
1292 / 100 is 12
```

Division and Type Casting

- Whenever a binary operator is applied to two **integers**, the result is an **integer**.
- Whenever a binary operator is applied to **at least one double**, the result is a **double**.
- What if we want the complete result even when we have two integers? Convert at least one operand to a double:
$$(double) n1 / n2$$

or

$$(double) n1 / (double) n2$$
- The conversion is accomplished by means of a **type cast**, which consists of a type name in parentheses.

The Pitfalls of Integer Division

Consider the following Java statements, intended to convert 100° Celsius temperature to its Fahrenheit equivalent:

```
double CelsTemp = 100.0;  
double FahrTemp = 9 / 5 * CelsTemp + 32;
```

How will the expression for **FahrTemp** be evaluated?

How can the problem be **solved**?

How could the problem have been **avoided** in the first place?

The modulus operator %

In Java, using the modulus operator results in the remainder when one integer is divided by another integer.

```
3 % 5    is 3
5 % 3    is 2
1292 % 100 is 92
```

An **error** will occur if an attempt is made to use the modulus operator with one or more floating point (double) values.

The modulus operator is useful in a surprising number of programming applications.

Precedence or Hierarchy of Operations

If an expression contains more than one operator, precedence rules are used to determine the **order of evaluation**.

From highest to lowest precedence:

- Expressions in **P**arentheses
- Unary minus ($-$), type casting
- **M**ultiplication and **D**ivision (left to right)
(includes the modulus operator $\%$)
- **A**ddition and **S**ubtraction (left to right)

Examples

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

$$\text{a) } 3 * 5 / 2 \quad \text{is} \quad 15 / 2 \quad \text{or} \quad 7$$

$$\text{b) } 3 / 8 \% 4 \quad \text{is} \quad 0 \% 4 \quad \text{or} \quad 0$$

$$\text{c) } 3 / (8 \% 4) \quad \text{is} \quad 3 / 0 \quad \text{or} \quad \textit{undefined - nan}$$

$$\text{d) } 5 - 2 * 3 \quad \text{is} \quad 5 - 6 \quad \text{or} \quad -1$$

$$\text{e) } 5 * 2 - 3 \quad \text{is} \quad 10 - 3 \quad \text{or} \quad 7$$

$$\text{f) } 5 * 2 / 3 \quad \text{is} \quad 10 / 3 \quad \text{or} \quad 3$$

$$\text{g) } 8 - 6 * 2 + 7 \% 2 \quad \text{is} \\ 8 - 12 + 7 \% 2 = 8 - 12 + 1 = -3$$

Examples of Mixed Mode Arithmetic

When both integer and double values are used in the same arithmetic expression, the partial result **remains an integer** until a floating point value is involved as an operand during the evaluation.

$2 / 5 / 4.0$ is $0 / 4.0$ or 0.0

$2 / (5 / 4.0)$ is $2 / 1.25$ or 1.6

$5.0 * 2 / 4$ is $10.0 / 4$ or 2.5

$5.0 * (2 / 4)$ is $5.0 * 0$ or 0.0

What is the value of:

$(1 + 2) \% 3 * 4 + 5 * 6 / 7 * (8 \% 9) + 10?$

Assignment Statements

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

- The value of a variable in a program can be changed by using an **assignment statement**, which has the general form:

```
variable = expression;
```

- The value of the expression on the right hand side is computed and assigned to the variable that appears on the left.
- The assignment statement: **total = total + tax;** adds the **current** values of the variables `total` and `tax`, then stores that sum back in the variable `total`.
- When a new value is assigned to a variable, the old value is lost.

Shorthand Assignments

- Statements such as `total = total + tax;` are so common that Java allows the following shorthand form:
`total += tax;`

- The general form of shorthand assignment is:

```
variable op= expression;
```

where *op* is any of Java's binary operators (+ - / * %).

- The effect of this statement is the same as:

```
variable = variable op expression;
```

To triple the value of salary: `salary *= 3;`

Auto-Increment and Decrement Operators

- Another common computation is to add or subtract one from an integer.
- Java provides several methods for **incrementing** and **decrementing**.

- To increment:

```
x = x + 1;
```

```
x += 1;
```

```
x++;
```

- To Decrement:

```
x = x - 1;
```

```
x -= 1;
```

```
x--;
```

Applications of Integer Arithmetic

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

Integer division and the **modulus operator** can be used

- to **group quantities together** or
- to **extract groups** from larger quantities.

Example

Suppose we have:

- several buckets of eggs,
- egg cartons (capable of holding 12 eggs each)
- we want to box the eggs so all cartons are full except perhaps the last one if we don't have a multiple of 12 eggs

Given the number of eggs, how do we calculate the **minimum** number of egg cartons needed to transport them safely?

Graphical Representation

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

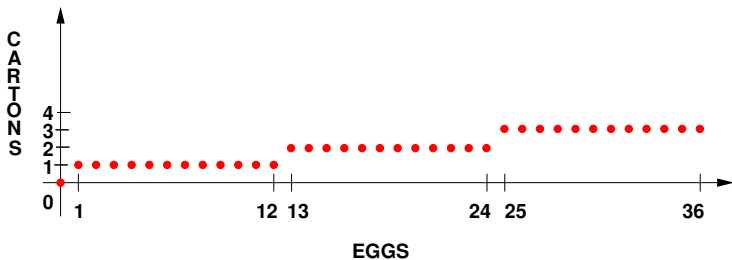
Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types



$12j - 11 \dots 12j$ eggs $\Rightarrow j$ cartons

j^{th} carton has 1 egg $\dots j^{\text{th}}$ carton is full $\Rightarrow j$ cartons

Discovering an Expression

Suppose there are x eggs. . . Let j be the **number of cartons**:

$$12j - 11 \leq x \leq 12j$$

$$(12j - 11) + 11 \leq x + 11 \leq 12j + 11$$

$$12j \leq x + 11 \leq 12j + 11$$

$$j \leq \frac{x + 11}{12} \leq \frac{12j + 11}{12}$$

$$j \leq \frac{x + 11}{12} \leq j + \frac{11}{12}$$

`int numberOfCartons = (numberOfEggs + 11) / 12;`

Furthermore...

Suppose we have a contract with a restaurant to supply eggs in cartons of **3** and **7** eggs each. To determine how to package the eggs:

```
// number of groups of 7
int numberOfSevens = numberOfEggs / 7;
```

```
// number of groups of 3 from eggs remaining
int numberOfThrees = (numberOfEggs % 7) / 3;
```

```
// 0, 1, or 2 eggs leftover after boxing the rest
int leftover = (numberOfEggs % 7) % 3;
```

Alternatively, Store Intermediate Values

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

```
// number of groups of 7  
int numberOfSevens = numberOfEggs / 7;
```

```
// number of eggs left after grouping by 7 (0 - 6)  
int remaining = numberOfEggs % 7;
```

```
// number of groups of 3 from eggs remaining  
int numberOfThrees = remaining / 3;
```

```
// 0, 1, or 2 eggs leftover after boxing the rest  
int leftover = remaining % 3;
```

Boolean Expressions

- The only values in the boolean domain are **true** and **false**
- The **boolean** type provides exactly the values needed in order for programs to **make decisions**.
- The **boolean** type is named for the English mathematician, George Boole, who introduced a system of logic now known as *Boolean algebra*, the foundation for the boolean data type.
- The **operators** used with the boolean data type fall into two categories: **relational** and **logical**.

Boolean Operators

- There are six **relational** operators that compare values of other types and produce a boolean result.

==	Equals	!=	Not equal
<	Less than	<=	Less or equal to
>	Greater than	>=	Greater or equal to

- There are also three **logical** operators, which are used with boolean operands:

&&	Logical AND	p&&q	— both p and q
	Logical OR	p q	— either p or q, or both
!	Logical NOT	!p	— the opposite of p

Notes on the Boolean Operators

- Remember that Java uses "=" to denote **assignment**. To test for **equality**, use the "==" operator.
- It is **not legal** in java to use more than one relational operator in a single comparison, as is often done in mathematics. To express the **compound inequality**
$$0 \leq x \leq 9$$
in Java, make both comparisons **explicit**, as in
$$0 \leq x \ \&\& \ x \leq 9$$
- The || operator means **either or both**
- Be careful when you combine the ! operator with && and || because the interpretation often differs from informal English.

Short-Circuit Evaluation

- The `&&` and `||` operators are evaluated using a strategy called **short-circuit** mode which evaluates the right operand only if it needs to do so.
- In `&&`, if the left operand is false, there is no need to evaluate the right operand because
false && anything
is always false
- In `||`, if the left operand is true, there is no need to evaluate the right operand because
true || anything
is always true

Designing for Change

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

- While it is necessary for you to write programs the compiler can understand, good programmers are equally concerned with writing code that *people* can understand.
- Programs must be maintained over their life cycle, hence human **readability** is important.
- Typically, as much as 90 percent of the programming effort comes *after* the initial release of a system.

Good Programming Style

There are several useful techniques that you can adopt to increase readability:

- Named constants enhance both readability and maintainability
- Proper indentation helps make program structure clear; use **ctrl-shift-F** to reformat statements in netbeans. You can also **right-click** in the editor and select **FORMAT** to do the same thing.
- The **purpose** of variables and methods should be clearly expressed in their names

Primitive Data Types

Java defines eight primitive types to represent simple data.

The programs in our text use only the following four:

`int` represents integers, which are whole numbers such as 17 or -53

`double`* represents numbers that include a decimal fraction, such as 3.14159265.

`boolean` represents a logical value, either **true** or **false**

`char` represents a single character

* Such values are called **floating-point** numbers; the name **double** comes from the fact that the representation uses twice the minimum precision.

Summary of Primitive Java Types & Operations

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

Type	Common Operations		
<code>byte</code>	The arithmetic operators:		
<code>short</code>	add(+)	subtract(-)	multiply(*)
<code>int</code>	divide(/)	remainder(%)	
<code>long</code>	The relational operators:		
	equal to (==)	not equal (!=)	less than (<)
	less or equal (<=)	greater than (>)	greater or equal (>=)
<code>float</code>	The arithmetic operators except %		
<code>double</code>	The relational operators		
<code>char</code>	The relational operators		
<code>boolean</code>	The logical operators:		
	&& (and)	(or)	! (not)

Summary of Primitive Java Types — Domains

A **data type** is defined by

- 1 a **set of values** and
- 2 a **set of operations**

Here are the Java primitive type domains:

Type	Size	Range of Values
<code>byte</code>	8-bit integers	-128 to 127
<code>short</code>	16-bit integers	-32,768 to 32,767
<code>int</code>	32-bit integers	-2,146,483,648 to 2,146,483,647
<code>long</code>	64-bit integers	-9,223,372,036,754,775,808 to 9,223,372,036,754,775,807

Java Primitive Types, Continued

Mat 2170
Chapter
Three

Java
Expressions

Week 3

Java
Expressions

Variable
Declarations

Operators

Mixed Mode

Assignment

Applications

Boolean
Expressions

Data Types

Type	Size	Range of Values
<code>float</code>	32-bit floating-point numbers	$\pm 1.4 \times 10^{-45}$ to $\pm 3.4028235 \times 10^{38}$
<code>double</code>	64-bit floating-point numbers	$\pm 4.39 \times 10^{-322}$ to $\pm 1.7976931348623157 \times 10^{308}$
<code>char</code>	16-bit characters encoded using Unicode	
<code>boolean</code>	the values <code>true</code> and <code>false</code>	