Mathematical Calculations in Java

Mrs. G. Chapman

Review of Declaring Variables

```
int sum;
```

typeidentifier

This statement declares sum.

It reserves a space in memory large enough to store an int.

Assignments Statements

sum = 120;

identifer assignment value

operator

We are initializing sum to 120.

We use the assignment operator (=) to give a value to a variable. **This is not used for comparison. (Does sum equal 120?)**

Variables- The value of a variable can be changed.

```
public class Geometry
{
public static void main (String [] args)
{
int sides = 7;
System.out.println (“A heptagon has “ + sides + “ sides.”);


sides = 10;
System.out.println (“A decagon has “ + sides + “ sides.”);

sides = 12;
System.out.println (“A dodecagon has “ + sides + “ sides.”);
}
}
```

In Memory:
sides
~~7~~
~~10~~
12

Output of previous code

```
Output:
A heptagon has 7 sides.
A decagon has 10 sides.
A dodecagon has 12 sides.
```

What is happening in memory?

When ever sides is assigned a new value, it writes over the previous value. So, when sides is assigned 10, it “forgets” the 7, and now holds the 10. This happens again when it is assigned 12… it “forgets” the 10 and now holds 12. Once it is assigned a new value, the old one is lost forever.

Assignment Statements

sides = 10;

The value on the **right** is given to the variable on the left.

This does not mean “equals” for comparison!!

The value that is assigned must be of the **same type.** sides = 5.5; This will generate an type mismatch error.

Arithmetic Expressions

Expressions: combinations of operators and operands to perform calculations.

The value calculated does not have to be a number, but often is.

Arithmetic Operators

int and double operators:
+ addition
-subtraction
* multiplication
/division
% modulus or remainder

Modulus Operator %

% (mod) remainder operator – returns the remainder when you divide 2 **integers or doubles**.

Example: int result = 7 % 3;

2

3 7

6

1 Remainder

result

1

2 Types of Division

1. integer division – when both operands are of type integer, we truncate the fractional part.

```
result = 7 / 3;
```

<u>result</u>
2

2. floating point division – if one or both operands are floating point numbers, then the fractional part is kept.

```
double dresult = 7.0 / 3.0;
```

<u>dresult</u>
2.333333333…

Order of Operations

Java has to follow the order of operations when computing an expression

Parenthesis
Multiplication, Division, Modulus
Addition, Subtraction, String Concatenation
Assignment

As a reminder if operators are on the same level, we work left to right.

Assignments work right to left.

+ with Strings

+ is the concatenation operator with Strings.

It is used to “add” Strings together. I put “add” in double quotes, because Strings are called immutable… meaning their value can never change. (Read next slide)

Strings are immutable

This means that Strings can never change. When using methods or operators on Strings, we aren’t changing Strings rather creating new String.

Recall: String class is very special. It is the only class that we can create objects without using the keyword “new”. When ever there is a String literal, or addition of Strings, etc. Strings are automatically created.

String addition

System.out.println (“A dodecagon has “ + sides + “ sides.”);

Working left to right… the String “A dodecagon has “ is being added with an int.

The integer is “promoted” to a String… a new String is created: “12” The 2 Strings are added together: “A dodecagon has 12”

Now we add this String with the String “ sides.”

“A dodecagon has 12 sides.”

This is a brand new String that is output to the user.

The Problem with Concatenation

```
System.out.println (“The value is “ + 24 + 25);
```

```
Output:
The value is 2425
```

Every time Java sees a String it promotes what is being added to also be a String… so…

When the first to pieces are added our result is a String: “The value is 24” 24 is no longer and int!!

When this is concatenated with 25, we get our output above.

Fixing the Problem

We need to use parenthesis:

System.out.println (“The value is “ + (24 + 25));

By using parenthesis, we force Java to add 24 and 25 first, resulting in 49.

Then we do our String addition… 49 is promoted to a String, and our result is….

```
The value is 49
```

Type Conversions

Sometimes we will need to change the type of a variable in order to use it with another variable or value.

Values can be widened or narrowed.

Widening: If you are going from a type that has a smaller memory storage to one with a larger memory storage, you are widening. int to double

Narrowing: If you are going from a type with a larger memory storage to a smaller memory storage, you are narrowing. double to int

*** When doing a Narrowing conversion, we run the risk of losing data.

Types of Conversions

There are 3 types of conversions:

1. Arithmetic Promotion
2. Assignment Conversion
3. Type Casting

Arithmetic Promotion

This is a widening conversion that will change the type so that arithmetic operations can be performed.

A perfect example is concatenating a String with an int.

Another example:

double dresult = 7.0 / 2;

2 gets promoted to a double for floating point division.

This type of conversion is done automatically, and does not require any intervention on the part of the programmer.

Assignment Conversion

If we assign an int to a double, we are converting it to a double.

```
int startingValue = 3;
double targetValue = startingValue;
```

The 3 stored in startingValue is changed to a double, and then assigned.

We can not go the other way around: startingValue = targetValue; This will generate a type mismatch error.

Type Casting

We can force a variable to change type by using typecasting. In front of the variable you wish to change, put the new type in parenthesis.

Example: double ans = (double)7 / 3;

This will change 7 into a double so that we are doing floating point division rather than integer division. ans = 2.33333333, rather than 2.0.

Typecasting a double an int

```
public class Money
{
private double amount;
public Money (double initialAmt)
{
amount = initialAmt;
}

public int dollarAmount()
{
return (int)amount;
}
}
```

By typecasting amount an int, we ***truncate*** (remove) the decimal portion of the number. It is NOT rounded.

Ex. if amount = 5.50; the value being returned by return (int)amount; is 5.