1.00 Lecture 4

Data Types, Operators

Reading for next time: Big Java: sections 6.1-6.4

Promotion

increasing capacity ->

Data Type	Allowed Promotions
double	*None*
float	double
long	float,double
int	long,float,double
char	*int,long,float,double*
short	int,long,float,double
byte	short,int,long,float,double

- Java performs promotions silently, from lower capacity types to higher capacity types, in operations and assignment (=)
- When doing binary operations, Java promotes byte or short to int
  - In all other cases it promotes the smaller to larger capacity
- Don't mess around: just use int (long sometimes) and double

Casting

- To convert a data type to a lower capacity type, you must cast it explicitly

```
long s= 1000*1000*1000;
int q=  (int) s;
```

- You are responsible for making sure the variable fits in the lower capacity representation
  - If it doesn t, you get no warning, and there is garbage in the variable (more shortly on this topic)
- You can cast variables into higher capacity types, if needed
  - In lecture 1, when computing the fraction grad students, you could have cast `int students` to `double`

```
double s2= (double) students;
```

Exercise

- Create a new project Lecture4
  - Write a class CastTest
  - In the main() method:
    - Declare `int`s x1=17, x2=20 and x3=12
    - Try to declare an `int 2x= 34`. What happens?
    - Compute the average of `x1, x2` and `x3`. Be careful.
    - Declare a `long big= 9876543210L;` (remember the `L`)
    - Try to set `int x4 = big` and print x4. What happens?
    - Cast `big` to an `int` and see what happens.

    If you have time:
    - Declare a `double small= 2.0 -0.000000000000001;`
      - `Enter number of zeros (14) exactly`
    - Try to set `int s= small`. What happens?
    - Cast `small` to an `int`. Is this ok?

Arithmetic operators

Table in precedence order, highest precedence at top

Operators	Meaning	Example	Associativity

++
--
+ (unary)
- (unary)
	increment
	decrement
	unary +
	unary –
		i= d++; x= ++q;
		--z; y= (a--) + b;
		c= +d;
		e= -f;
			Right to left

*
/
%
	multiplication
	division
	modulo
		a= b * c * d;
		e= f / g;
		h= i % j;
			Left to right

+ -	addition subtraction	k= m + n + p; q= s – t;	Left to right

% operator defined only for integers

Arithmetic operator exercise

- Create a class ArithmeticTest in Lecture4
- Write a main() method in class ArithmeticTest
  - Set the number of 1.00 students to 136
  - Increment this by one, then decrement by one
    - Easy come, easy go, before add or drop date
  - Set the number of 1.001 students to 20
  - Find total students (1.00, 1.001), but increment the 1.001 students by one first, all in one line
  - If we put students in groups of three, how many groups of three are there?
  - How many students are left over?
  - Use the debugger to see your answers
    - Don't write any System.out.println statements

Precedence, associativity

- **Operator precedence is the order in which operators are applied. Do exercises on paper:**
  - **Operators in same row have equal precedence**

```
int i=5; int j= 7; int k= 9; int m=11; int n;
n= i + j * k - m;              // n= ?
```

- **Associativity determines order in which operators of equal precedence are applied**

```
int i=5; int j= 7; int k= 9; int m=11; int n;
n= i + j * k / m - k;          // n= ?
```

- **Parentheses override order of precedence**

```
int i=5; int j= 7; int k= 9; int m=11; int n;
n= (i + j) * (k - m)/k;        // n= ?
```

Operator exercises

- **What is the value of int n:**
  - `n= 1 + 2 - 11 / 3 * 5 % 4;` `// n= ?`
  - `n= 6 + 5 - 20 / 3 * 7 % 4;` `// n= ?`
  - `int i= 5; int j= 7; int k= 9;`
  - `n= 6 + 5 - ++j / 3 * --i % k--;` `// n= ?`
  - `i= 5;`
  - `n= i + ++i;` `// n= ?`
  - `// Don't ever do any of these!`

Integer arithmetic properties

- **Overflows occur from:**
  - **Division by zero, including 0/0 (undefined)**
    - **Programmer has responsibility to check and prevent this**
    - **Java will warn you (by throwing an exception) if it can t do an integer arithmetic operation**
  - **Accumulating results that exceed the capacity of the integer type being used**
    - **Programmer has responsibility to check and prevent, as in zero divides**
    - **No warning is given by Java in this case**

Floating point exercise

- **Write a program to solve the following:**
  - **You have a 1 meter long bookshelf**
  - **There are things that are 0.1m, 0.2m, 0.3m, 0.4m and 0.5m long**
  - **Starting with the smallest thing, place each on the bookshelf until you can t place any more**
  - **How many things can you put on the shelf?**
  - **How much space is left over?**
- **Download BookshelfTest0, which has the skeleton of the code**
  - **This exercise demonstrates the imprecision of floating point numbers**
  - **Java approximates the real number line with 2^{64} integers**

Floating point exercise

```
public class BookshelfTest0 {

   public static void main(String[] args) {
        double lengthLeft= 1.0;       // Remaining space
        int booksPlaced= 0;           // Books on shelf so far
        double length= 0.1;           // Length of book

        // Your code here: try to place books of length 0.1, 0.2,
        // 0.3, 0.4, 0.5m on shelf. Loop while enough space

        System.out.println("Books placed: "+ booksPlaced);
        System.out.println("Length left: "+ lengthLeft);
   }
}
```

Floating point problem

- **How do we fix this?**
  - **Never use `if (a == b)` with floats or doubles**
    - **== is ok with integer types**
  - **Always use `if (Math.abs(a – b) < TOLERANCE)`**
    - **Where `TOLERANCE` is about 10^{-6} float or 10^{-15} double**
    - **Or a variation on this if the operator is not ==**
    - **Add `TOLERANCE` to one side or the other in an inequality to accommodate the representation error**
- **Correct the previous exercise**

MIT OpenCourseWare
http://ocw.mit.edu

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.