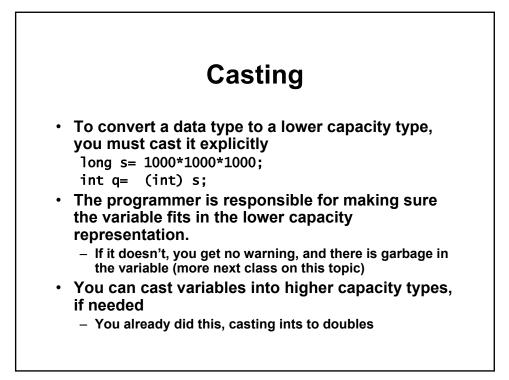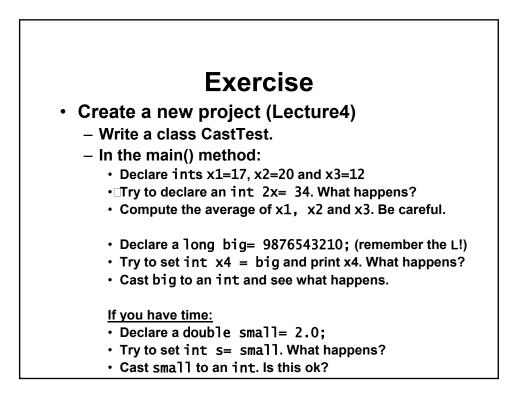# 1.00 Lecture 4

Data Types, Operators

Reading for next time: Big Java: sections 6.1-6.4 (still)

Promotion

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

increasing capacity →

- Java does 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;
  ```
- **The programmer is 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 next class on this topic)
- **You can cast variables into higher capacity types, if needed**
  - You already did this, casting ints to doubles

Exercise

- **Create a new project (Lecture4)**
  - **Write a class CastTest.**
  - **In the main() method:**
    - Declare `ints` 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= 9876543210;` (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;`
    - Try to set `int s= small`. What happens?
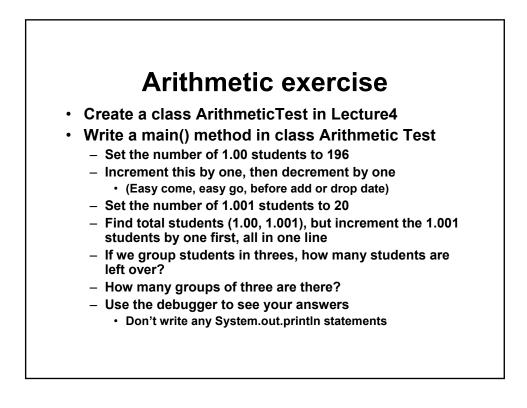   - Cast `small` to an `int`. Is this ok?

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 (int)
		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

---

Arithmetic exercise

- **Create a class ArithmeticTest in Lecture4**
- **Write a main() method in class Arithmetic Test**
  - **Set the number of 1.00 students to 196**
  - **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 group students in threes, how many students are left over?**
  - **How many groups of three are there?**
  - **Use the debugger to see your answers**
    - **Don't write any System.out.println statements**

Precedence, Associativity

- **Operator precedence is in the order of the earlier table. 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 (discussed later)**
  - **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**

Integer Arithmetic Exercise

```java
public class IntArithmetic {
    public static void main(String[] args) {
        int cubicInch= 2000*1000*1000;
        System.out.println("cubicInch: " + cubicInch);
        cubicInch += cubicInch;  // Add more space
        System.out.println("cubicInch: " + cubicInch);
    }
}
```

Integer Arithmetic Exercise

```java
public class IntArithmetic {
    public static void main(String[] args) {
        int cubicInch= 2000*1000*1000;
        System.out.println("cubicInch: " + cubicInch);
        cubicInch += cubicInch;  // Add more space
        System.out.println("cubicInch: " + cubicInch);
    }
}
```

```
// Output
cubicInch: 2000000000
cubicInch: -294967296
```

It's necessary to analyze the range of your results, under worst case circumstances. You often use a long to hold sums of ints, etc.

• Exercise: fix this method

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?**

Floating Point Exercise

```
public class BookshelfTest {

    public static void main(String[] args) {
        double lengthLeft= 1.0;
        int booksPlaced= 0;

        // Your code here: things are 0.1, 0.2, 0.3, 0.4, 0.5m
        // Loop while there is still enough space for next item

        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
  - 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 ==
- Correct the previous exercise