CSC 142

Primitive types

[Reading: chapter 5]

Identifiers: review (1)

- class, method or variable names
- Java is case sensitive
  - class HelloWorld ≠ class helloWorld
- An identifier is composed of
  - letters, digits, underscores, currency symbol
  - can't start with a digit
  - 1_Bad, thisIsOK, this_is_also_OK_123
  - can't be a reserved java keyword

```
private String class; //Error
```


Identifiers: review (2)

- Style
  - class names: capitalize the first letter of each word in the name, e.g.

    class ThisIsMyClass
  - method and variable names: same except for the first letter which is lowercase, e.g.

    myMethod, myVariable
  - constants (we will see how to declare them)

    use capital letters and separate the words with an underscore, e.g. MAX_VALUE

Types in Java

- Every variable and method has a type
- 2 categories
  - primitive types, for simple values that have built-in functionalities in the language. An example is an integer value.
  - reference (or class) types, for objects.

Primitive types (1)

- int for an integer value, e.g. 1, -2 but not 1.334
  - when coding, write

```
int i;
i = 2342;
```

- double for a floating point value, e.g. 1.33, -134.2E-10
  - when coding, write

```
double x;
x = -2.234;
```


Primitive types (2)

- char for a single keyboard character, e.g. 'a', '$', but not 'sara'
  - when coding, write

```
char c;
c = '@'; /* use single quotes to write a
  character explicitly in your program */
```

  - some special characters
    - '\t' for tab, '\n' for new line, …
- boolean for the logical value true or false
  - when coding, write

```
boolean isSunday;
isSunday = false;
```

- There are other primitive types (see later)

final keyword

- Sometimes, a variable should not change (e.g. a mathematical constant such as pi)
- To make a variable constant, use final when declaring it

```
final double PI=3.1416;
...
PI = 4.;  // Error
          // PI is a constant
```

- A constant must be initialized when declared

```
final int DAYS_IN_A_WEEK;
DAYS_IN_A_WEEK = 7;
//Error! write final int DAYS_IN_A_WEEK =7;
```


What about static final?

- Any instance of a class has its own instance fields.

```
public class Calendar{
 public final int NUMBER_OF_MONTHS=12;
 // etc...
}
```

- All Calendar instances have their own copy of the instance field NUMBER_OF_MONTHS.
- Better to have one copy shared by all of the instances: use the static keyword.

```
public static final int NUMBER_OF_MONTHS=12;
```

- We will see other uses of static later

Doing Arithmetic

- Usual arithmetic operations:
  - unary +,- (e.g. –3 or +2)
  - binary *, /, + and – (e.g. 2*3)
- Between integers, / is the integer division
  - **27/12 is 2 (not 2.25)**
- % (mod operator): a%b is the remainder of the division of a by b, e.g.
  - 27%12 is 3 since $12 \overline{)27}$ with quotient 2, remainder 3
  - **49%5 is 4**
  - **2%3 is 2**
  - **6.2%2.9 is 0.4**

An example

- How many hours, minutes and seconds in 6789 seconds?
- The java answer

```
int seconds=6789;
int hours=seconds/3600;
seconds=seconds%3600;
int minutes=seconds/60;
seconds=seconds%60;
System.out.println("6789s="+hours+"h"+
minutes+"mn"+seconds+"s");
//prints 6789s=1h53mn9s
```


Order of precedence

- What is a+b*c in Java?
  - Is it a + (b*c) or (a+b)*c?
  - Of course, we expect a+(b*c)
- Java uses the following precedence rules
  - evaluate expressions in parentheses first (start with the innermost set of parentheses)
  - to evaluate an expression, do unary –, then do *,/ and %, and then do binary + and -
- Don't hesitate to use parentheses to clarify your arithmetic expressions.

Associativity

- What is a/b*c in Java?
  - Is it a / (b*c) or (a/b)*c?
  - Of course, we expect (a/b)*c
- Java binary arithmetic operators associate left to right within the same level of precedence.
- Note: not all operators associate left to right
  - What about the assignment =?
- Add parentheses to the following expression to show how it is evaluated
  - a+b/-c*d-e
  - answer: (a + ((b/(-c))*d))-e

Shortcuts

- op= where op is +,-,*,%, /

```
x+=a; // same as x=x+a;
x%=b; // same as x=x%b;
```

- post increment and decrement: ++ and --

```
x++; // post increment: same as x=x+1;
x--; // post decrement: same as x=x-1;
```

- Also pre increment and decrement: ++ and --

```
++x; //x=x+1; or --x; //x=x-1;
```

  - Difference?
    - post: the variable is updated after being used
    - pre: the variable is updated before being used
    - Same if used in isolation. Not the same in an expression or assignment

```
int i=0, j=0, a, b;
a=i++; // a is 0 and i is 1
b=++j; // b is 1 and j is 1
```


Conversions in arithmetic

- Java automatically converts types in obvious cases, e.g.

```
double x = 3.2;
int i = 8;
System.out.println("x/i="+x/i);
// 3.2/8 is computed as 3.2/8.0 which is 0.4
```

- Be careful

```
System.out.println(1/2*3.4); // prints 0!
```

- When in doubt, don't let java decide. Java allows only the most obvious conversions:

```
int i = 3.0; //Nope! compilation error
```

- Use a cast instead

Casting

- Consider

```
double x = 3.8;
int i = x;  // Error!
```

- Convert to an int using a cast, i.e. write

```
int i = (int)x; //i is 3 (drop fractional part)
```

- Java is strongly typed. It only performs safe automatic conversions (int to double…). Other conversions are the programmer's responsibility.
- Syntax for casting

```
sometype var;
var = (sometype)expression;
/* not always allowed. The compiler will tell
you (e.g. boolean b = (boolean)3.4; is an
error) */
```


All java numerical primitive types

- integers: int is the most common
  - in order of increasing capacity:

```
byte (1 byte: range = -2^7 to 2^7-1)
short (2 bytes: range = -2^15 to 2^15-1)
int (4 bytes: range = -2^31 to 2^31-1)
long (8 bytes: range = -2^63 to 2^63-1)
```

- floating point variables: prefer double
  - in order of increasing capacity:

```
float (4 bytes absolute value=1.4x10^-45 to
 3.4x10^38)
double (8 bytes absolute value=4.9x10^-324 to
 1.8x10^308)
```


Precision

- Java allocates a finite amount of memory to store a floating point variable
  - the value may be truncated. Don't always expect exact results

```
double x = 1.34*6.24; //=8.3616
// Use the DecimalFormat class to control
// the display of numbers (in java.text.*)
DecimalFormat d;
// 16 digits after the decimal point
d = new DecimalFormat("0.0000000000000000");
System.out.println("x="+d.format(x));
// prints x=8.3616000000000010
```

- A computer works in base 2. A number may have a few number of digits in base 10 and a large number in base 2. If so, the number is truncated.

Math class

- A class that defines mathematical constants and functions, e.g. (check documentation)
  - PI, E, sqrt, cos, sin, tan, exp, log…

```
// area of a circle
double area = Math.PI*radius*radius;
// square root of a positive number
System.out.println("Square root of 21 is "+
 Math.sqrt(21));
// arc tangent of a number
double piOver4 = Math.atan(1);
```

- Don't need to instantiate the Math class to use it
  - All members of the Math class are declared static : more on this later

Examples using Math

- If x is a double, find the long i that is the closest to x (e.g. if x is 3.9, i is 4 or if x is –2.2, i is –2)
  - Without the Math class

```
if (x>=0)
  i = (long)(x+0.5);//(long)(3.9+0.5) is 4
else
  i = (long)(x-0.5);//(long)(-2.8-0.5) is -3
```

  - With the Math class

```
i = Math.round(x);
```

- generate a random integer >=10 and <20

```
(int)(Math.random()*10+10);
//random returns a random double >=0 and < 1
```


More about char and boolean

- booleans: boolean (1 bit true or false)
- characters: char (2 bytes Unicode)
  - stored on 2 bytes
  - each char has its own code (2 byte number)
    - 'A' is 65, 'a' is 92, '@' is 64
    - allows to compare characters ('b'<'m' is true)
  - 2 bytes is large enough to allow the coding of most characters appearing in all human languages (2^{16}=65536)

Primitive types as instance fields

- An instance field is automatically initialized to a default value (of course, the value can then be changed by any method of the class).
  - null if it has a class type
  - what if it has a primitive type?
    - 0 for numbers (int, double…)
    - '\0' for char
    - false for boolean

```
public class SomeClass{
  private int x; //x initialized to 0
   // more code...
}
```


Wrapper classes

- What if we want to treat a primitive type as an object?
  - use a type Wrapper
  - classes Integer, Double, Character, Boolean, etc…

```
int someInt = 32;
Integer myInt = new Integer(someInt);
```

  - Use: some methods always take an object as input (e.g. in Vector class). Can't use them with a primitive type. But OK with a wrapper class. See later when dealing with collections.

Boxing / Unboxing

- Since JDK 1.5, Java can automatically cast a primitive type to its associated wrapper type (boxing), or cast a wrapper type to its associated primitive type (unboxing).

```
Integer i = 3; // boxing
int k = i; // unboxing
```

- Convenient for algorithms that require reference types.
- Boxing / unboxing blurs the distinction between primitive types and reference types, but doesn't eliminate it.