04 Working with Primitive Data

[Close all sections] Only visible sections will be included when printing this document.

References: L&L 2.1, 2.2, 2.3, 2.4, 2.6

▽ Programs model something real

In a program we use *data* to model real-world things. Objects in the real world can be described by numbers, letters, text, and statements that are true or false. These different kinds of data have different *types*.

A data type determines what values can be represented (numbers, integers, text, images, etc.) and what operators can be used to change it. For example, numbers support arithmetic operators, strings (of text) can be joined together or split apart, truth values can be combined and tested.

Statements and Expressions

A *statement* is a single instruction to the computer, e.g.:

```
System.out.println("Hello");
myTurtle.penDown();
```

An *expression* is anything that can be evaluated to produce a single value

- `1 + 1` evaluates to the integer 2
- `"Hello"` evaluates to the String Hello
- `2` evaluates to the integer 2 (obvious though that is)

Java primitive types we can use to model the real world

There are eight primitive data types in Java, four of which we'll use frequently:

- Four represent integers: `byte`, `short`, `int`, `long`
  - `int` should be your default choice
- Two represent floating point (real/decimal) numbers: `float`, `double`
  - `double` should be your default choice
- One represents individual characters (see Appendix C): `char`
- One of them represents Boolean values: `boolean`

The value of a primitive type is stored directly in the variable. We'll see later how objects are stored in memory differently.

Why so many different kinds of numbers?

The different integer and real-number data types take up differing amounts of space in memory (and consequently can represent different ranges of values). While improvements in memory availability and processor speed have reduced the need for the smaller data types, the language still supports them to give programmers a choice (and even nowadays memory *may* be limited, such as in an embedded or mobile device… or a robotic rover on Mars!).

Integer type	bits	Minimum value	Maximum value
byte	8	–128	127
short	16	–32,768	32,767
int	32	–2,147,483,648	2,147,483,647
long	64	–9,223,372,036,854,775,808	9,223,372,036,854,775,807

The range of the real-number types is better understood in terms of their precision for really small or really large values (that is, how many digits after the decimal point, or how many before it). They can represent these values as either positive or negative. The values below are approximate.

Real number type	bits	Smallest positive value	Largest positive value
float	32	1.4×10^{-45}	3.4×10^{38}
double	64	4.9×10^{-324}	1.8×10^{308}

Value literals

Expressions that literally represent a single value

- `int` literals are represented by digits with no decimal point, such as `1` or `1024`
  - To indiciate an integer literal is a `long` value, append `L`, as in `1L`, `2L` or `1024L`
- `double` literals are represented by digits that include a decimal point, such as `1.`, `1.0` or `1.5`
  - These literals may also be given using 'scientific notation' such as `1.5e3`, which is equal to 1.5×10^3, or 1500.0.
  - To indicate that a floating-point number is actually a float, append `f` or `F`, as in `1.0f` and `2.5F`
- `char` literal values are represented by a character in single quotes, such as `'a'`, `'Z'` or `'1'` (which is the character that *looks* like the digit 1, but is not the *value* 1).
  - Some special characters, like newline and tab have escape sequences: `'\n'` and `'\t'` each represent *one* character.
- `boolean` literals are `true` and `false` (all lower case with no quote marks)

What about Strings?

Strings are not primitives, they are objects. But they are used so frequently that they also have a literal representation: `"text within double quotes represents a String object"`.

▽ Variable declaration

A *variable* is a name that refers to a location in memory. A variable must be declared before use by specifying the variable's name (its *identifier*) and the *type* of information that will be held in it. When planning a program, a 'plain language' (pseudocode) way of writing this is:

```
Variables:
type identifier, short description
```

For example, to plan for an integer to represent your age in years:

```
Variables:
int myAge, age in years
```

The equivalent syntax in Java is:

- **type identifier** ;
  e.g.: `int age; //age in years`
  or with initialisation
- **type identifier** = **expression** ;
  e.g.: `int height = 175;`

Rules and guidelines for identifiers

- The name of a variable is
  - Called an *identifier*
  - Chosen by the programmer (you)
- Syntax: can contain only letters, digits, underscore (_) or dollar sign ($), but cannot begin with a digit.
  - So no spaces allowed

Scope

Scope is the set of locations in a program where an identifier is visible.

In general, a variable is visible within its enclosing *block* (pair of braces `{}`) and below/after its declaration.

Primitive object wrappers

Each primitive type has an equivalent class type that allows the primitive to be used where an object is needed (most ways of storing large amounts of data in Java work only with objects) and has utility methods to working with that kind of primitive data. The classes are (you don't need to remember them):

Primitive type	Class type
`byte`	Byte
`short`	Short
`int`	Integer
`long`	Long
`float`	Float
`double`	Double
`char`	Character
`boolean`	Boolean

▽ Variable assignment

Assignment changes the value of a variable by assigning it the value of an expression

Pseudocode to use when planning could be any of the following:

```
identifier becomes expression value
or
identifier = expression value (where you should read = as 'becomes')
or
identifier ⟵ expression value
```

For example, to state that the variable myAge is assigned the value 18, you could write any of the following:

```
myAge becomes 18
```
```
myAge = 18
```
```
myAge ⟵ 18
```

The Java syntax for this is then:

- **identifier** = **expression** ;
  e.g.: `myAge = 18;`

The type of the expression and type of the variable must be *compatible*. That is, they must either be the same or the variable type must be a superset of the expression. For instance, you can assign an `int` value to a `double` because the set of real numbers includes all integers.

Constants

A constant is a named value (i.e., a variable) whose value, once assigned, cannot be changed during program execution. Value usually assigned when declared.

In psuedocode:

```
Constants:
type IDENTIFIER = expression, short description
```

as in:

```
Constants:
int WORK_DAYS = 5, assuming Monday to Friday work week
```

The Java syntax for declaring a constant is:

- `final` **type IDENTIFIER** = **expression** ;
  or, if going to assign value (once!) later
- `final` **type IDENTIFIER** ;

The **naming convention** is to write the identifier in UPPER_CASE with words separated by underscores (_), unlike the camel case used for other Java names.

Examples:

```
final double WORK_HOURS_PER_DAY = 7.5; //9am-5pm with 30 minute lunch break
final boolean DEBUG_MODE = true; //debugging messages will be printed if true
```

Advantages of using constants are:

- Gives names to otherwise unclear literal values, so makes code more readable
- Facilitates simple changes to the code; value may be used in many places but on needs to be changed in one
- Prevents inadvertent errors from mistyping a literal value in some places or forgetting to update it in all places.

Arithmetic Expressions

- An expression is a combination of operators and operands.
- Arithmetic expressions compute numeric results and make use of the arithmetic operators:
  - Addition `+` (e.g., `13 + 4` is 17)
  - Subtraction `–` (e.g., `13 – 4` is 9)
  - Multiplication `*` (e.g., `13 * 4` is 52)
  - Division `/` (e.g., `13 / 4` is 3)
  - Remainder (modulo) `%` (e.g., `13 % 4` is 1)

Integer division and numerical type conversion

If either or both operands to an arithmetic operator are floating-point (`double`), the result is a floating-point (`double`).

If both operands to division (`/`) have type `int` then the operation is *integer division* and the result is also an `int`. So, `3 / 4` evaluates to 0, but `3.0 / 4` evaluates to 0.75.

Type casting to treat integers as floating-point

Sometimes you will need to tell the compiler that a value of one data type should be converted to a different data type. The syntax for this is `(type to convert to)expression to convert`, as in `(double)2`, which tells the compiler to convert the integer value 2 to a double (2.0).

At this stage of your programming career the most likely cases you will need this are:

1. Converting an `int` to a `double` in order to perform real-valued division instead of integer division. For example, if you want to calculate the average length of words in a sentence, you would calculate the sum of the lengths of the words (an integer, call it `sum`) and then divide it by the total number of words (also an integer, call it `count`). Because an average should include a fractional part you need to tell the compiler to perform real-valued division, as in

   ```
   double averageLength = (double)sum / count;
   ```

2. Truncating a real-valued `double` value so you can store it (without the fractional part) in an `int` variable. This is less common, but there will be occasions when you have performed some real-valued arithmetic but only require the whole number component (equivalent to the mathematical operation floor). As long as you know the value in the double is no larger than the maximum value of `int` then this is safe:

   ```
   double real = 3.1415;
   int floor = (int)real; //floor will contain the value 3
   ```

   There is also a built-in function, `Math.floor(double a)`, but it returns another `double` (without its fractional component), so doesn't help you store the result in an `int` variable.

Activity: Evaluate the following expressions involving `int`s and `double`s and work out the type of the result and its value:

1. `1/2`
   Show Solution
2. `(double)1/2`
   Show Solution
3. `1/2.0`
   Show Solution
4. `(int)(1/2.0)`
   Show Solution
5. `(int)1/2.0`
   Show Solution

Operator Precedence

Operators can be combined into complex expressions. Precedence determines the order in which they are evaluated. Precedence is:

- Parentheses ()
- Multiplication, division, and remainder
- Addition, subtraction

(operators with the same precedence are evaluated from left to right)

Activity: What is the order of evaluation in the following expressions?

- a + b * c - d / e
- a * b + c / d - e
- a / (b + c) - d % e
- a / (b * (c + (d - e)))

Show Solution

Activity: Imagine the following code is placed inside the `main` method of a Java program (between `public static void main(String[] args) {` and the closing `}`). Expressions 3 and 4 are taken from above. What do you expect the output to be? (Hint: evaluate each expression independently.)

```
int a = 5;
int b = 4;
int c = 3;
int d = 2;
int e = 1;
int expr1, expr2, expr3, expr4;

expr1 = a * b + c;
expr2 = a + b * c;
expr3 = a / (b + c) – d % e;
expr4 = a / (b * (c + (d – e)));

System.out.println("Expression 1 is " + expr1);
System.out.println("Expression 2 is " + expr2);
System.out.println("Expression 3 is " + expr3);
System.out.println("Expression 4 is " + expr4);
```

Show Solution

☑ Test Yourself: Declarations and Assignment

Activity: Suppose that the following statements were placed in the `main` method of a new Java program. What would happen when the program is executed? Answer the questions below.

```
1 int a, b, c;
2
3 a = 10;
4 b = 8;
5 c = a + b;
6 a = c / b;
```

What value does each variable have after each of these lines has been executed? (Create a table like the one below, or take a copy of the code and write down the values next to each line.)

Line	a	b	c
3	Show Solution	Show Solution	Show Solution
4	Show Solution	Show Solution	Show Solution
5	Show Solution	Show Solution	Show Solution
6	Show Solution	Show Solution	Show Solution

Show All Solutions

(Note that this reveals or hides *all* solutions in this document.)

Activity: Suppose the following statements were placed in the `main` method of a different Java program. What would happen as the program is executed?

```
1 final int PRIME = 31;
2 final int DIVISOR = 3;
3 int a, b;
4
5 a = PRIME;
6 b = 2;
7 a = (a + b) / DIVISOR;
```

What value does each variable after each of the indicated lines has been executed?

Line	PRIME	DIVISOR	a	b
1	Show Solution			
2	Show Solution	Show Solution		
5	Show Solution	Show Solution	Show Solution	
6	Show Solution	Show Solution	Show Solution	Show Solution
7	Show Solution	Show Solution	Show Solution	Show Solution

After lines 1 and 2, can the values of `PRIME` and `DIVISOR` ever change?
Show Solution

▽ More About Strings and Keyboard Input

Type Conversion and Strings

Every primitive type has a corresponding String representation, e.g.:

- `int` 1 is represented by the String `"1"`
- `double` 1.0 is represented by the String `"1.0"`
- `char` `'a'` is represented by the String `"a"`
- `boolean` true is represented by the String `"true"`

An expression mixing primitives and Strings will convert the primitive to its corresponding String

For example, `"Adam " + 12` results in `"Adam 12"`

Reading keyboard input with the `Scanner` class

- Outside your program class, import Scanner
  - `import java.util.Scanner;`
- Create a Scanner to read from 'standard input' (mostly this is the keyboard):
  - `Scanner sc = new Scanner(System.in);`

To read and store…

- …an `int`:
  - `int myInt;`
  - `myInt = sc.nextInt();`
- …a `double`:
  - `double myDouble;`
  - `myDouble = sc.nextDouble();`
- …a `boolean`:
  - `boolean myBool;`
  - `myBool = sc.nextBoolean();`
- …a String:
  - `String myString;`
  - `myString = sc.next();`

☑ Test Yourself: Arithmetic and Formatted Messages

Activity: Suppose the following lines of code were placed in the `main` method of a Java program (and were the only contents of the `main` method). What would the output of the program be?

```
int height = 176;   //height in cm
double mass = 67.4; //mass in kg
double bmi;

bmi = mass/Math.pow(height/100.0, 2);

System.out.print("Calculating BMI using height of " + height + " cm ");
System.out.println("and weight of " + mass + " kg.");
System.out.println("BMI: " + bmi);
```

Show Solution

▽ Challenge Activity

If you're feeling comfortable with the material covered so far consider attempting this optional activity.

Activity: Write an algorithm (in English or pseudocode) to reverse the digits of a 3-digit number `x`.

Assume that `x` is already declared as an `int` within the range 100–999 and has already been assigned a value, e.g., 123. Your algorithm should declare any additional variables you need and, at the end of the algorithm, `x` should contain the reverse of its original digits.

Hint: you will need to use / (integer division), % (remainder), and * (multiplication), and will need at least one (and probably three) additional variables.

Once you think you've got it right, translate it into Java code to test it out.