Handout 2

Basic Java Constructs: Variables, Primitive Types, Interactive Input/Output.

Problem: According to the Guinness book of World Records:

Les Stuart from Midjimba Beach, Australia, spent 16 years typing the numbers from 1 to 1,000,000 on his typewriter. He started his effort in 1982 and finished in 1998, used 19,990 sheets of paper and exhausted 7 typewriters.

Tai Ping Adorro would like to beat Les’s record and needs an estimate of how many numbers Les typed daily.

Calculate the average amount of numbers Les typed each day. (1984, 1988, 1992 and 1996 were leap years).

Can you calculate how many digits per day Les typed on average?

Design: specify

Input:
Output:
Data:

Algorithm:
A Java program that makes the required calculation.

/
* Program calculates typist's average daily speed
 * (in numbers per day) based on how many full years it took
 * to type numbers 1 to 1,000,000
 *
*/

public class TypistsAverage {
    public static void main (String args[]) {
        // declare VARIABLES to store number of years and leap years
        int numYears, numLeapYears;
        // declare a CONSTANT: how many numbers to type
        final int numNumbers = 1000000;
        // prompt the user to enter number of years spent typing
        System.out.println("Please enter how many full years (would be) spent typing ");
        numYears = SavitchIn.readLineInt();
        // prompt the user to enter how many leap years within that period
        System.out.println("How many of these years are leap years? ");
        numLeapYears = SavitchIn.readLineInt();

        int days; // days in numYears years
        double average; // daily average
        days = 366*numLeapYears + 365*(numYears-numLeapYears);
        average = numNumbers/days;

        // print out the result
        System.out.println("The average rate is " + average + " numbers per day");
    }
}

Let's go over parts of this program, simultaneously introducing Java basic concepts.
1. Program starts with comments:
   - any number of lines enclosed within /* */ block, or
   - starting with // to the end of line
Comments are necessary for readability – element of good programming style

2. Code to begin the program

```java
public class TypistsAverage {
    public static void main(String[] args) {
    }
}
```

Java applications all have similar code at the beginning.

First line  `public class TypistsAverage` declares a class.
Explanation:
- `public` - is an access control keyword that means can be used by other classes. For now all classes we define are going to be public.
- `class` is a keyword that must be followed by the name of the class.
- `TypistsAverage` is the name of the class – you get to choose it
- **File name must be the same as the name of the class it defines.**

Next: definition of method `main`:

```java
public static void main(String[] args) {
}
```

**definition of class TypistsAverage is enclosed within { } that follow**

**definition of the main method of class TypistsAverage**

- `public` - similar to above
- `static` - will have to wait for explanation
- `void` - returns no value.
- `main` - reserved method name; `main` method of a class is executed first when a class is executed.
- `String[] args` – a list of parameters passed to the program through a command line – we will not be using it for a while.

---

1 Keyword – (a.k.a. reserved word) is a word that has a special meaning in a language and cannot be used for other purposes.
3. Start a program with instructions to the user on what the program does and/or how to use it:

   System.out.println("Please enter how many full years (would be) spent typing ");

   Screen output methods:
   System.out.println()
   System.out.print()

4. Variable declarations:

   int numYears, numLeapYears;

   and later in the program

   int days;   // days in numYears years
   double average; // daily average

Variable - is a named location for data storage
- It can hold only one type of data
- for example only integers, only floating point (real) numbers, or only characters
- All program variables must be declared before using them. A variable declaration associates a name with a storage location in memory and specifies the type of data it will store: e.g. int, float, double, char, String

| numYears   |   |
| numLeapYears |   |
| days       |   |
| average    |   |

Java has Primitive types and Class types

**Primitive types** (e.g. char, int, float, double, long, boolean)
- the simplest types, they define values that cannot decompose into other types (in other words are atomic)

Each primitive type has operators that apply to values of that type:
  e.g. numeric operations, boolean operators.
Used to form expressions.
Class types (e.g. SavitchIn, String)
- more complex, used to define objects
- composed of other types (primitive or class types)
- contain both data and methods that objects of class can perform.

To **set** a variables to store a given value – use **assignment** operator =

Not the same as in algebra, It means -
“Assign the value of the expression on the right side to the variable on the left side.”

Can have any expression on the right hand side of =
Restriction: the type of the variable must be compatible with the type of the expression on the right hand side.

**Examples:**

```java
// initialize and later decrement the variable
int count = 10; // initialize counter to ten
count = count - 1; // decrement counter

days = 366 * numLeapYears + 365 * (numYears - numLeapYears);
```

```java
//assign the value that was read from keyboard input:
umYears = SavitchIn.readLineInt();
```
5. **Constants**, e.g.

```java
// declare a CONSTANT: how many numbers to type
final int numNumbers = 1000000;
```

- are similar variables, but cannot be changed.
- declared with the keyword `final`

Advantages of using named constants:

*Example*: use `MORTGAGE_INTEREST_RATE` instead of 8.5

- Easier to understand program because reader can tell how the value is being used
- Easier to modify program because value can be changed in one place (the definition) instead of being changed everywhere in the program.
- Avoids mistake of changing same value used for a different purpose

6. **More on types and type conversion**:

Recall that the type of the variable must be compatible with the type of the expression on the right hand side. So, would the following be legal?

```java
double x;
int n = 5;
x = n;
```

The answer is yes. The integer value of `n` is *cast* to a double 5.0, then assigned to `x`. This is called *implicit* casting because it is done automatically.

The following chart describes the allowable automatic type conversion (a.k.a. type casting):

<table>
<thead>
<tr>
<th>Type</th>
<th>Convertible to</th>
<th>Automatically convertible</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>byte</code></td>
<td><code>short</code></td>
<td><code>int</code></td>
</tr>
<tr>
<td><code>short</code></td>
<td><code>int</code></td>
<td><code>long</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>long</code></td>
<td><code>float</code></td>
</tr>
<tr>
<td><code>float</code></td>
<td><code>double</code></td>
<td></td>
</tr>
</tbody>
</table>

The logic behind it: can automatically convert so long as the new type uses more or same amount of storage.

Otherwise, need to explicitly specify the conversion: e.g.

```java
int n;
double x = 2.89;
n = (int)x;  //legal in java. n will be set to 2.
```
7. **Characters**

Type char values define single printable characters. They are actually stored in memory as integer numbers according to a special code: each printable character (letter, number, punctuation mark, space, and tab) is assigned a different integer code.

- the codes are different for upper and lower case
- for example 97 may be the integer value for ‘a’ and 65 for ‘A’

Java uses Unicode (Unicode includes all the ASCII codes plus additional ones for languages with an alphabet other than English)

Casting a char value to int produces the ASCII/Unicode value

**Problem:** what would the following display?

```java
char answer = 'y';
System.out.println(answer);
System.out.println((int)answer);
```

8. **Keyboard input using SavitchIn** (see also chart on p.95):

```java
class SavitchIn has methods to read values of different types. They are

SavitchIn.readLineInt() -- returns an int value
SavitchIn.readLineLong() -- long
SavitchIn.readLineFloat() -- float
SavitchIn.readLineDouble() -- double
SavitchIn.readLineNonwhiteChar() -- char
SavitchIn.readLine() -- String
```
9. **Rules of good programming style.**

Look at the following program and try to understand what it does.

```java
public class TA {
    public static void main (String args[]) {
        int a, b;
        double x, y;
        final int c = 1000000;
        System.out.println("Please enter how many full years (would be) spent typing and how many of these are leap years");
        a = SavitchIn.readLineInt(); b = SavitchIn.readLineInt();
        x = 366*a + 365*(a-b); y = c/x;
        System.out.println("The average rate is " + y + " numbers per day");
    }
}
```

To make programs readable:

- Document program with comments.
- Use meaningful names for variables, constants, classes, etc.
- Use indentation and line spacing as shown in the examples in the text
- Always include a “prologue” (an brief explanation of the program at the beginning of the file)

*Created by T. Babaian 9/8/02*