Software Construction
Basics of Java Part II

Dhammika Elkaduwe

Department of Computer Engineering
Faculty of Engineering

University of Peradeniya

ILOs

- data types
- ranges
- type conversion when doing arithmetic
- arrays (how to define, length)
- scope of variables
- masking variables
- *for loops*

Variables

There are three type of variable in Java:

- Local variables (within a function)
- Class variables (static variables)
- Instance variables (non-static variables)

Syntax to define a variable:

```
<data type> <identifier1>, <identifier2>;
```

Example:

```
int a, b;
char grade = 'A'; // grade variable is defined and initialise
String hello = "Hello World", bye = "Good Bye World";
String combined = hello + ", " + bye; // concatenated and assign.
```

see Variables.java

Identifiers

Following rules apply when giving variable names:

- **Variable names are case-sensitive (so are all Java identifiers)**
- **Cannot use Java keywords as variable names (Example: public, for)**
- **Can begin with a letter, $ sign or _**
  - Typically the $ is reserved for auto-generated variables
  - So, regular variables begins with a letter or _
- **Unlimited number of letters, digits, $ signs and _ can be used**
- If the variable name is a single word, use all simple letters.
- If the variable name has more than one word, capitalise the first letter of each subsequent word.
- If the variable stores a constant value, use all capital letters separating words with _

Note: **bold fonts** indicate rules, other indicate convention.

Java keywords

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

taken from http://www.tutorialspoint.com/java/java_tutorial.pdf

Primitive data types

Following primitive data types are available in Java:

Data type	Usage	Representation	Range
byte	whole numbers	8bit, 2's compliment	-128 to 127
short	whole numbers	16bit, 2's compliment	-32,768 to 32, 767
int	whole numbers	32bit, 2's compliment	-2^{31} to $2^{31}-1$
long	whole numbers	64bit, 2's compliment	-2^{63} to $2^{63}-1$
float	fractions	32bit, IEEE 754	-
double	fractions	64bit, IEEE 754	-
boolean	logic values	true/false	-
char	Unicode	16bit, Unicode	-

Precision, memory usage, speed

What should be the output of the following code?

```
float floatNum = 102.2f; // f means convent to float
double d = 1.2d; // by default factions are considered as double
System.out.println(floatNum + d);
```

see Variables.java

Note:

- Some numbers cannot be accurately represented (fractional part)
- Approximate the value and store
- Conversion from one type to other will also lose some precision.
- Using *double* will require more memory.

```
$ java Variables
...<some other output>
103.39999694824219
$
```

Division and types

```
int a = 8;
int b = 3;
float c = 3.0f;// or float c = 3f;
double d = 3d; // or double d = 3.0;
System.out.println("int (10) /int (3) \t= " + (a/b));
System.out.println("int (10) /float (3)\t= " + (a/c));
System.out.println("int (10) /double (3)\t= " + (a/d));
```

see TypeConversion.java

- both numbers integer $\Longrightarrow$ integer division (no fractional part)
- at least one number is float/double $\Longrightarrow$ division has a fraction.
  - precision would depend on the type.
  - double has more precision than float
- by default, Java takes fractional numbers as doubles (example: 3.0 is double. 3.0f is *forced* to float)

Note: types of variables

Java variables can be

- Local variables (defined inside a function)
- Static variables (defined for the class)
- Non-static variables

All these are defined in the same way.
The difference is in their behaviour.

Scope of a variable

```
class Scope {
  static int i = 0;
  static int MAX = 10; // by convension constants are capitalised
  public static void main(String [] args) {
    int i = 10; // mask the static int i, defined above
    System.out.println("i = " + i );
    for(int j=0; j < MAX; j++) // no MAX in function. Take
        static one
       System.out.println("j = " + j);

    // j = 10; //will not work. j's scope is within the for loop
  }
}
```

- *Visible within the block in which it is defined.*
- *Can mask a variable in the class or object*

Arrays

```
public static void main(String [] args) { // arrays of Strings
    called args
  for(int i=0; i < args.length; i++)
     System.out.printf("%s\n", args[i]);

  int [] a = {1, 2, 3, 4, 5};
  for(int i=0; i < a.length; i++)
      System.out.printf("a[%d] = %d\n",i, a[i]);

    int [] b = new int[10]; // similar to b = malloc(sizeof(int)
        * 10)
    // new will create a new array object
    for(int i=0; i < b.length; i++) b[i] = b.length - i;
    for(int i=0; i < b.length; i++)
       System.out.printf("b[%d] = %d\n",i, b[i]);

}
```

Defining arrays: example 2

```
<data type> [] <identifier>;
```

```
<data type> [] <identifier> = {val1, val2}
```

```
<data type> [] <identifier> = new <data type> [ELEMENTS];
```

```
String [] ar = {"CO225: Software Construction",
  "CO224: Computer Architucture",
  "CO226: Database Systems"};

for(int i=0; i < ar.length; i++)
   System.out.println(ar[i]);
```

see StringArray.java

Arrays: Things to note

```
public static void main(String [] args) { // arrays of Strings
    called args
  ....
  int [] a = {1, 2, 3, 4, 5};
  ...
  int [] b = new int[10]; // similar to b = malloc(sizeof(int) *
      10)
  ...
}
```

Things to note:

- How arrays are passed as arguments to functions
- How a new array can be created
  - as a static array (values given at the start)
  - as dynamic array (using *new* key word
- Arrays are passed by reference (not by value)

ILOs: Revisited

- data types
- ranges
- type conversion when doing arithmetic
- arrays (how to define, length)
- scope of variables
- masking variables
- *for loops*