Data Types, Variable and Operator

Outline of the Presentation

- ✓ Pre requisite (Do it yourself)
- ✓ Identifiers and naming Convention
- ✓ Data types and Variables
- ✓ Type conversion and casting, type promotion

1.Concepts of OOPs

(definition and examples of abstraction, classes, objects, encapsulation, inheritance and polymorphism)

2.Keywords

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

true, false, and null are not keywords, just like literal value

3. Operators – Arithmetic Operators

Operator	Result
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

3. Operators – Bitwise Operators

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

3. Operators – Relational Operators

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

3. Operators – Boolean Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

3. Precedence of Operators

Highest			
( )	[ ]	.	
++	– –	~	!
*	/	%	
+	–		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|			
&&			
||			
?:			
=	op=		
Lowest			

4. Control Statements

1. Selection Statements
    - (a) if else
    - (b) nested ifs
    - (c) if-else-if ladder
    - (d) switch-case
2. Looping Statements
    - (a) while
    - (b) do-while
    - (c) for --- (comma operator)
    - (d) for-each (to be discussed in arrays)
3. Jumping Statement
    - (a) break
    - (b) continue
    - (c) return

5. Lexical issues

(literals, identifiers, separators, comments)

Identifiers and Naming Conventions

Names of things that appear in the program are called identifiers.

Rules:

✓An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).

✓An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.

✓An identifier cannot be a reserved word.

✓An identifier cannot be true, false, or null.

✓An identifier can be of any length.

*Java is case sensitive

*Do not name identifiers with the **$** character. By convention, the **$** character should be used only in mechanically generated source code

*Descriptive identifiers make programs easy to read

Names of things that appear in the program are called identifiers.

Rules:

- ✓ An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).
- ✓ An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- ✓ An identifier cannot be a reserved word.
- ✓ An identifier cannot be true, false, or null.
- ✓ An identifier can be of any length.

Which of the following identifiers are valid?

applet, Applet, a++, —a, 4#R, $4, #44, apps

Conventions (not rules)

✓Use lowercase for variables and methods. If a name consists of several words, concatenate them into one, making the first word lowercase and capitalizing the first letter of each subsequent word—for example, the variable **radius** and the method **computeArea()**.

✓Capitalize the first letter of each word in a class name—for example, the class name Circle

✓Capitalize every letter in a constant, and use underscores between words—for example, the constants **PI** and **MAX_VALUE**

* Do not choose class names that are already used in the Java library. For example, since the **Math** class is defined in Java, you should not name your class **Math**.

* Avoid using abbreviations for identifiers. Using complete words is more descriptive. For example, **numberOfStudents** is better than **numStuds**, **numOfStuds**, or **numOfStudents**

Datatype and Variables (do it yourself)

1. Declaring Variables
   datatype variablename;
   datatype variablename 1, variablename 2, …. variablename n;
2. Initializing Variables
   datatype variablename = value;
3. Defining Constants
   final datatype CONSTANTNAME = VALUE;

Data types

1. Numeric
2. Character
3. Boolean

1. Numeric datatypes

Name	*Range*	*Storage Size*
`byte`	-2^7 (−128) to 2^7-1 (127)	8-bit signed
`short`	-2^{15} (−32768) to $2^{15}-1$ (32767)	16-bit signed
`int`	-2^{31} (−2147483648) to $2^{31}-1$ (2147483647)	32-bit signed
`long`	-2^{63} to $2^{63}-1$ (i.e., −9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: −3.4028235E + 38 to −1.4E−45 Positive range: 1.4E–45 to 3.4028235E + 38	32-bit IEEE 754
`double`	Negative range: −1.7976931348623157E + 308 to −4.9E−324 Positive range: 4.9E − 324 to 1.7976931348623157E + 308	64-bit IEEE 754

2. Character datatype

```
char letter = 'A';
char numChar = '4';
```

- ✓Java supports Unicode
- ✓65,536 characters possible in a 16-bit encoding
- ✓16-bit Unicode takes two bytes.
- ✓ The range of a char is 0 to 65,536. There are no negative chars.
- ✓The standard set of characters known as ASCII still ranges from 0 to 127 as always.
- ✓The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character

.

2. Character datatype

✓Escape Sequences for special characters

Character Escape Sequence	*Name*	*Unicode Code*
\b	Backspace	\u0008
\t	Tab	\u0009
\n	Linefeed	\u000A
\f	Formfeed	\u000C
\r	Carriage Return	\u000D
\\	Backslash	\u005C
\'	Single Quote	\u0027
\"	Double Quote	\u0022

3. Boolean datatype

✓The boolean data type is used to declare Boolean variables.

✓A boolean variable can hold one of the two values: true and false

✓true and false are literals

Automatic type promotion in expressions

Consider the following line of code:

```
byte b = 50;
 b = b * 2;
```

Consider the following line of code:

```
byte b = 50;
 b = b * 2;          // Error! Cannot assign an int to a byte!
```

The code is attempting to store 50 * 2, a perfectly valid byte value, back into a byte variable. However, because the operands were automatically promoted to int when the expression was evaluated, the result has also been promoted to int.

Thus, the result of the expression is now of type int, which cannot be assigned to a byte without the use of a cast.

1. Numeric Type Conversions

Consider the following statements:

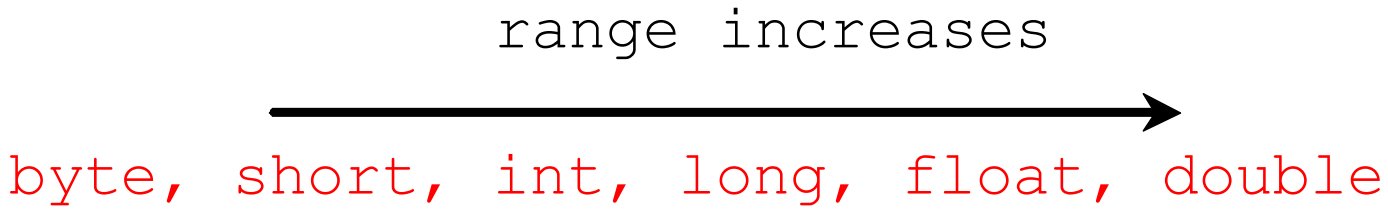
```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. All byte, short, and char values are promoted to int.
2. if one operand is a long, the whole expression is promoted to long.
3. If one operand is a float, the entire expression is promoted to float.
4. If any of the operands is double, the result is double.

1. Numeric Type Conversions

The result of 1/2 is 0 but result of 1.0/2 is 0.5



Casting converts a value of one data type into a value of another data type

* widening a type – 1. Casting a variable of a type with a small range to a variable of a type with a larger range
  2. can be performed automatically (implicit casting)
* narrowing a type – 1. Casting a variable of a type with a large range to a variable of a type with a smaller range
  2. must be performed explicitly (explicit casting)

Note: Not all the types are compatible, Ex. boolean is not compatible with numeric or char types.

1. Numeric Type Conversions

The result of 1/2 is 0 but result of 1.0/2 is 0.5

Implicit casting

double d = 3; (type widening)

Explicit casting

int i = (int)3.0; (type narrowing)

int i = (int)3.9; (Fraction part is truncated)

What is wrong? int x = 5 / 2.0; It would work if int x = (int)(5 / 2.0);

2. Character Type Conversions

- ✓ A **char** can be cast into any numeric type, and vice versa
- ✓ When an integer is cast into a char, only its lower 16 bits of data are used; the other part is ignored

```
int i = 65;
char ch = (char)i;
System.out.println(ch); // ch is character A
```

- ✓ When a floating-point value is cast into a char, the floating-point value is first cast into an int, which is then cast into a char

```
char ch = (char)65.25; // decimal 65 is assigned to ch
System.out.println(ch); // ch is character A
```

- ✓ Implicit casting can be used if the result of a casting fits into the target variable. Otherwise, explicit casting must be used.

```
int i = 'a';                    //correct
byte b = i;       // incorrect   byte b = (byte) i; //correct
```

2. Character Type Conversions

✓All numeric operators can be applied to char operands. A char operand is automatically cast into a number if the other operand is a number or a character. If the other operand is a string, the character is concatenated with the string.

```
class Test
    { public static void main(String [] ar)
            {
                int i = '2' + '3';                          // (int)'2' is 50 and (int)'3' is 51
                System.out.println("i is " + i);            // i is 101
                int j = 2 + 'a';                            // (int)'a' is 97
                System.out.println("j is " + j);            // j is 99
                System.out.println(j + " is the Unicode for character "+ (char)j);
                System.out.println("Chapter " + '2');
              }
     }
```

- O/P is
  - i is 101
  - j is 99
  - 99 is the Unicode for character c
  - Chapter 2

```
class Promote
 {
public static void main(String args[])
{
byte b = 42;
char c = 'a';
short s = 1024;
int i = 50000;
float f = 5.67f;
double d = .1234;
double result = (f * b) + (i / c) - (d * s);
System.out.println("result = " + result);
}
}
```