BINUS UNIVERSITY

GREATER JAKARTA • BANDUNG • MALANG

People
Innovation
Excellence

COMP6048001
Data Structures

Primitive Data Type and Reference Variable
Week 2

Maria Seraphina Astriani
seraphina@binus.ac.id

Session Learning Outcomes

Upon successful completion of this course, students are expected to be able to:

- LO 1. Describe the use of various data structures
- LO 2. Apply appropriate operations for maintaining common data structures
- LO 3. Apply appropriate data structures and simple algorithms for solving computing problems
- LO 5. Explain the efficiency of some basic algorithms

Topics

- Primitive Data Type
- Reference Variable

Review – Week 1 Topics

- **menti.com** from mentimeter.com

Primitive Data Type and Reference Variable

Primitive Data Type and Reference Variable

- Java distinguishes between two kinds of entities: primitive types (numbers, characters) and objects.
- Values associated with primitive-type data are stored in primitive-type variables.
- Objects, however, are associated with reference variables, which store an object's address.

Primitive Data Types

Java Primitive Data Types in Increasing Order of Range

Data Type	Range of Values	Size
`byte`	–128 through 127	1 byte
`short`	–32,768 through 32,767	2 bytes
`int`	–2,147,483,648 through 2,147,483,647	4 bytes
`long`	–9,223,372,036,854,775,808 through 9,223,372,036,854,775,807	8 bytes
`float`	Approximately $\pm 10^{-38}$ through $\pm 10^{38}$ and 0 with 6 digits precision	4 bytes
`double`	Approximately $\pm 10^{-308}$ through $\pm 10^{308}$ and 0 with 15 digits precision	8 bytes
`char`	The Unicode character set	2 bytes
`boolean`	`true`, `false`	1 bit

- The primitive data types for Java represent numbers, characters, and boolean values (true, false).
- Integers are represented by data types byte, short, int, and long; real numbers are represented by float and double.

Type `char`

- Type **`char`** is used in Java to represent characters. Java uses the Unicode character set (two bytes per character), which provides a much richer set of characters than the ASCII character set (one byte per character) used by many earlier languages.

- The table on the next slide shows the first 128 Unicode characters, which correspond to the ASCII characters.

Tutorial 1 - ASCII

- Input from user
- Print the ASCII value based on the input

```
Enter a character: f
ASCII value of f is: 102
```

From the ASCII table...

Symbol	Decimal	Binary
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010

Symbol	Decimal	Binary
a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
e	101	01100101
f	102	01100110
g	103	01100111
h	104	01101000
i	105	01101001
j	106	01101010
k	107	01101011
l	108	01101100
m	109	01101101
n	110	01101110
o	111	01101111
p	112	01110000
q	113	01110001
r	114	01110010
s	115	01110011
t	116	01110100
u	117	01110101
v	118	01110110
w	119	01110111
x	120	01111000
y	121	01111001
z	122	01111010

https://medium.com/@aidafarihabaharunsuratman/did-someone-actually-use-ascii-table-in-java-programming-9710a65c6ed9

The First 128 Unicode Symbols

	000	001	002	003	004	005	006	007
0	Null		Space	0	@	P	`	p
1			!	1	A	Q	a	q
2			"	2	B	R	b	r
3			#	3	C	S	c	s
4			$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v
7	Bell		'	7	G	W	g	w
8	Backspace		(	8	H	X	h	x
9	Tab		)	9	I	Y	I	y
A	Line feed		*	:	J	Z	j	z
B	Escape		+	;	K	[	k	{
C	Form feed		,	<	L	\	l	|
D	Return		-	=	M	]	m	}
E			.	>	N	^	n	~
F			/	?	O	_	o	delete

Tutorial 2 - Unicode

- Print Unicode Symbols

```
Unicode: A_*
```

Primitive-Type Variables

- Java uses declaration statements to declare and initialize primitive-type variables.

```
int countItems;
double sum = 0.0;
char star = '*';
boolean moreData;
```

Primitive-Type Constants

- Java programmers usually use all uppercase letters for constant identifiers, with an under- score symbol between words.

- The keywords **static final** identify a constant value that is **static** (more on this later) and **final**—that is, can't be changed.

```
static final int MAX_SCORE = 999;
static final double G = 3.82;
```

Operators

- The arithmetic operators (*, /, +, –) can be used with any of the primitive numeric types or type char, but not with type boolean.

- This is also the case for the Java remainder operator (%) and the increment (++) and decrement (–) operators.

Compound assignment = https://www.geeksforgeeks.org/compound-assignment-operators-java/
https://www.javatpoint.com/operator-shifting

Operator Precedence

Rank	Operator	Operation	Associativity
1	[]	Array subscript	Left
	()	Method call	
	.	Member access	
	++	Postfix increment	
	--	Postfix decrement	
2	++	Prefix increment	Right
	--	Prefix decrement	
	+ -	Unary plus or minus	
	!	Complement	
	~	Bitwise complement	
	(*type*)	Type cast	
	new	Object creation	
3	*, /, %	Multiply, divide, remainder	Left
4	+	Addition or string concatenation	Left
	-	Subtraction	
5	<<	Signed bit shift left	Left
	>>	Signed bit shift right	
	>>>	Unsigned bit shift right	
6	<, <=	Less than, less than or equal	Left
	>, >=	Greater than, greater than or equal	
	instanceof	Reference test	
7	==	Equal to	Left
	!=	Not equal to	
8	&	Bitwise and	Left
9	^	Bitwise exclusive or	Left
10	|	Bitwise or	Left
11	&&	Logical and	Left
12	||	Logical or	Left
13	?:	Conditional	Left
14	=	Assignment	Right
	*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, |=	Compound assignment	

Postfix and Prefix Increment

- In Java you can write statements such as

```
i = i + 1;
```

using the *increment operator*:

```
i++;
```

The postfix increment (or decrement) is more common.

This form is the *postfix increment*.

You can also use the *prefix increment*

```
++i;
```

Postfix and Prefix Increment

- What is the difference between `i++` and `++i`?

Postfix and Prefix Increment

- When the postfix form is used in an expression (e.g., x * i++), the variable i is evaluated and then incremented.

```
z = i++;
```

i is incremented, but z gets the value i had before it was incremented. So if i is 3 before the assignment statement, z would be 3 and i would be 4 after the assignment. In the assignment statement

- When the prefix form is used in an expression (e.g., x * ++i), the variable i is incremented before it is evaluated.

```
z = ++i;
```

i is incremented and z gets its new value, so if i is 3 before the assignment, z and i would both be 4 after the assignment statement.

Code example: z++ and ++z

Type Compatibility and Conversion

- In operations involving mixed-type operands, the numeric type of the smaller range is converted to the numeric type of the larger range.
- This means that if an operation involves a type int and a type **`double`** operand, the type **`int`** operand is automatically converted to type **`double`**.
- This is called a *widening conversion*.

Type Compatibility and Conversion

- In an assignment operation, a numeric type of a smaller range can be assigned to a numeric type of a larger range; for example, a type **`int`** expression can be assigned to a type **`float`** or **`double`** variable.
- Java performs the widening conversion automatically.

```
int item = . . . ;
double realItem = item;     // Valid - automatic widening
```

- However, the converse is not true.

```
double y = . . . ;
int x = y;    // Invalid assignment
```

Exercise 1

```
Line
1   public class Exercise2_1 {
2       public static void main(String[] args)
3       {
4           long a = 6;
5           int i = a;
6           i = i * a;
7           System.out.println(i);
8       }
9   }
```

I need your help to explore first, before I explain the whole things

1. Use “Exercise2_1.java” code
2. Please explain why it produced error? (It won’t compile)
3. Fix it (please keep `long` and `int` types, do not change them)

Type Compatibility and Conversion

- Why this code is invalid?

```
double y = . . . ;
int x = y;   // Invalid assignment
```

Other case & explanation

Type Compatibility and Conversion

- This statement is invalid because it attempts to store a real value in an integer variable.

- It would cause the syntax error possible loss of precision; double, required: int. This means that a type int expression is required for the assignment.

Other case & explanation

Type Compatibility and Conversion

- How to fix it?

```
double y = . . . ;
int x = y;   // Invalid assignment
```

- Hint: explicit *type cast* operations

Other case & explanation

Type Compatibility and Conversion

- You can use explicit *type cast* operations to perform a *narrowing conversion* and ensure that the assignment statement will be valid.

- In the following statement, the expression (int) instructs the compiler to cast the value of y to type **`int`** before assigning the integer value to x.

```
int x = (int) y;   // Cast to int before assignment
```

Other case & explanation

Recall – Print ASCII

- Still remember how to print ASCII?

- You also may write the following code to convert from character to ASCII

```
int a = (int)'A';
System.out.println(a);
```

Referencing Objects

- In Java, you can declare reference variables that can reference objects of specified types.
- For example, the statement

```
String greeting;
```

  declares a reference variable named greeting that can reference a String object.

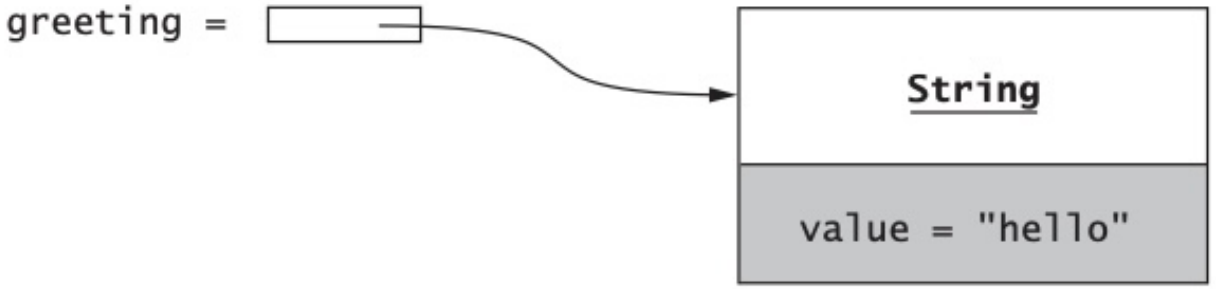
- The statement

```
greeting = "hello";
```

  specifies the particular String object to be referenced by greeting: the one that contains the characters in the string literal "hello".

Referencing Objects

- What is actually stored in the memory cell allocated to greeting is the address of the area in memory where this particular object of type String is stored.
- We illustrate this in the following figure by drawing an arrow from variable greeting to the object that it references (type String, value is "hello").



Creating Objects

- The Java **`new`** *operator* can be used to create an instance of a class.

- The expression

```
new String("qwerty")
```

  creates a new String instance (object) that stores the character sequence consisting of the first six characters of the top row of letters on the standard keyboard (called a “qwerty” keyboard).
- The expression `new String("qwerty")` invokes a special method for the String class called a *constructor*.

Creating Objects

- A constructor executes whenever a new object of any type is created; in this case, it initializes the contents of a String object to the character sequence "qwerty".

- The object created by the expression `new String("qwerty")` is an anonymous or unnamed object.
- Normally we want to be able to refer to objects that we create.
- We can declare a reference variable of type String and assign this object to the reference variable:

```
String keyboard = new String("qwerty");
```

FYI
Tutorial 3 – String Manipulation

- `length();`
- `toLowerCase();`
- `charAt(...);`

Input:

```
String x = "aBCDe";
```

Output:

```
5
D
abcde
```

Exercise 2 - Case

- You work at a private detective office and need to create an application that able to encrypt the text to a secret code.

- The length of the text should be **min. 5 letters**.
- All the text need to be **converted into lowercase** letters.
- The **1st** letter is **not ecrypted**.
- **2nd and 3rd** letters are **swapped**.
- The rest of the letters (4th letter until the end) need to be **jump over** some letters depend on the **previous** letter value.
  - The letter need to be re-started from a if reach z
- Example:
  - aBcdA → acbge
  - aBcdE → acbgi
  - Bcbzy → bbcby

FYI

a = 1, b = 2, c = 3, d = 4, e = 5, f = 6, g = 7, h = 8, i = 9, j = 10, and so on

4th letter = 3 (taken from the value of c) + 4 (the value of d) = 7 is g

5th letter = 4 (taken from the value of d) + 1 (the value of A) = 5 is e

Exercise 2 - Case

- Result

```
Welcome to Detective Encrypt Program
Type something >= 5 chars and press Enter:
asdf
Type something >= 5 chars and press Enter:
gotoOffice
gtoidulolh
```

1 a	2 b	3 c	4 d	5 e	
6 f	7 g	8 h	9 i	10 j	
11 k	12 l	13 m	14 n	15 o	
16 p	17 q	18 r	19 r	20 t	
21 u	22 v	23 w	24 x	25 y	26 z

Hint

	Letters	Value		Results
1st	g	7	*same*	g
2nd	o	15	*swap*	t
3rd	t	20	*swap*	o
4th	o	15	**20**+15=35	i
5th	O	15	**15**+15=30	d
6th	f	6	**15**+6=21	u
7th	f	6	**6**+6=12	l
8th	i	9	**6**+9=15	o
9th	c	3	**9**+3=12	l
10th	e	5	**3**+5=8	h

References

- Koffman, E. B., & Wolfgang, P. A. (2021). Data structures: abstraction and design using Java (4th Edition). John Wiley & Sons. [DSA]
- Weiss, M. A. (2010). Data Structures and Problem Solving Using Java (Fourth Edition). Addison-Wesley. [DSPS]
- Weiss, M. A. (2014). Data structures and algorithm analysis in Java (3rd Ed). Pearson.
- Karumanchi, N. (2017). Data Structures and Algorithms Made Easy In JAVA. CareerMonk.
- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data structures and algorithms in Java (6th Ed.). John Wiley & Sons.

Thank you