
G10S1 AP CSA Midterm Review

Ana, Dennis, Jett, Justin

2022-11-2

Contents

Lesson 1-5	5
- Java Program Template	5
Class:	5
Method	5
Constructor	6
Main Method	6
- Error & Exception	6
Compile-Time Error / Syntax Error	6
Run-Time Exception	7
Logic Error	7
- Output	7
- Built-in(Primitive Types)	8
Types	8
强转	8
Storage	9
Binary Number	9
- Variable&Constants	10
Identifier	10
Variable	10
Constants	10
- Expression	11
Types	11
Arthematic Expression	11
Truncation - int/int	12
String Concatenation	12
Relational Operators	12
Compound Assignment Operators	13
Increment and DEcrement Operators	13
Logical Operators	14
PRIORITY - IMPORTANT	14
Short Circuit	14
- De Morgan's Law	14
- Decision-Making Control Structures	15
If Statements	15
If-else Statements	15

Lesson 6-9	16
Lesson 6	16
Rewrite of nested if statement	16
Defusion of the compound logic in if statement	17
Example: loop year	17
Lesson 7	18
Lesson 8	19
-For loop	19
-Comparison between while & for loop	20
-Infinite for loop	21
-off by one error	21
Lesson 9	21
Nested loops	21
Lesson 10-12	23
Lesson 10	23
the basic introduction of object-oriented programming	23
what are the components of a class	23
Instance variables	24
Static variables:	24
public and private modifiers	24
constructor	25
This.	25
Create object in runtime	25
Methods	26
encapsulation	26
invoking instance methods in client class	27
static method:	27
Lesson 11	27
instance method invocation	27
Static method invocation	27
tostring	28
Parameter passing during the method invocation	29
object reference type	29
Method overload	29
Wrapper classes	29
Autoboxing	30
Autoboxing:	30

Lesson 12	30
String initialization 1	31
String initialization 2	31
“==” operator	31
Equals method(boolean)	31
Length method(int)	32
substring method(string)	32
indexOf method(int)	32
Compareto(string)	32
Appendix	32
Resources	32
Online Java IDEs	32
Offline Java IDEs	33
Java resources	33
Credits	33

Lesson 1-5

10-3 蒋韵涵 Ana

- Java Program Template

Class:

```
1 //template
2 [public/private] + class + [Classname]{
3
4 }
5
6 //Eg
7 public class Kumo{
8     /*Code*/
9 }
```

Classname 开头字母建议大写

Method

```
1 //template
2 [public/private] + [static/null] + [int/void/etc.(ReturnType)]
3     + [Methodname] + (parameter list) + {
4
5 }
6
7 //Eg.
8 public static void Kumo(String csa, double ap){
9
10 }
```

1. **access specifier:** public/private, 决定是否能在其他 class(client class) 中直接使用此 method
2. **static:** 添加与否决定 method 是属于 Object 还是整个 Class(static or instance)
 1. 不写 static 就是 instance
 2. Eg1. private int a = 0;
 - private 对应 access specifier - 代表这个 int 无法在其他 class 直接使用
 - 没有写 static 就代表是 instance
 - 所以 a 是一个 private instance

3. Eg2. 创建一个 Class Car

- 在 Car class 中, `change_price()` 就会是 instance method, 因为每个车子价格都会不一样, 所以我们在改价格时也是改每个 object(每个车) 各自的价格, 因而 method 不属于整个 Car Class
- 但是如果你很牛, 你要改变世界, 比如把世界上所有汽车都改成八个轮子的, 那么此时 `change_wheels()` 就是 static method, 因为这个改变是会改变所有属于 Car 类型的东西
- (详见后面 Class&Object 部分)

3. **ReturnType:** 返回类型, primitive type/object, void 就不返回 (可以理解为只是做了一个动作)

4. **parameter:** take-in

Constructor

- 创建一个属于 Class 的 object

```
1 //template
2 public [Classname]{
3
4 }
5
6 //Eg.
7 public BankAccount{
8     /* Code */
9 }
```

创建了一个 *BankAccount* 的 object

Main Method

```
1 public static void main(String[] args){
2     /* Code */
3 }
```

- Error & Exception

Compile-Time Error / Syntax Error

Sth is wrong according to the rules of language and the compiler finds it.

Eg. `system.out.print("a");` -> `System.out.print("a");`

Run-Time Exception

Errors detected by the program during execution, which causes the program to terminate before finishing execution

Eg.

Exception	Example/Meaning
ArithmeticException	division by 0
NullPointerException	method call for an object whose value is null
ArrayIndexOutOfBoundsException	int[] kumo = {1,2,3}; int wrong = a[4];
IndexOutOfBoundsException	和上面同理
StringIndexOutOfBoundsException	和上面同理

Logic Error

Error that causes the program to produce an unexpected output

不影响程序正常运行

- Output

- `System.out.print()` & `println()`
- Use `\` in front of signs to include signs into the String literal - for signs that are specific(eg. `" "`).

```
1 //Eg.
2
3 System.out.print("Kumo is \"happy\". ")
4 //Output: Kumo is "happy".
```

- Escape Sequence

Escape Sequence	Meaning
<code>\b</code>	backspace
<code>\t</code>	tab
<code>\n</code>	newline
<code>\r</code>	carriage return
<code>"</code>	double quote
<code>'</code>	single quote
<code>\</code>	backslash

- Comment
 - `//` : for a single current line
 - `/* ... */` : multi-lines

- Built-in(Primitive Types)

Start with a **lower case** letter

Types

- `int`: $-2^{31} \sim 2^{31}-1$
- `double`
- `boolean`

强转


```

1 //Eg.
2
3 int a = 1;
4 double b = (double)a;
5
6 // (type) variable

```

Storage

- 1 byte = 8 bits
- int : 4 bytes
- double: 8 bytes
- boolean: 1 bit
- 注意 *Overflow - result out of range* , 因为有 storage 限制 - 属于 Logic Error

Binary Number

■ Unsigned Binary Representation

$$1111_{(2)} = 1 \cdot 2^3_{(10)} + 1 \cdot 2^2_{(10)} + 1 \cdot 2^1_{(10)} + 1 \cdot 2^0_{(10)} = 15_{(10)}$$

Most Significant Bit (MSB)

Least Significant Bit (LSB)

General Rule:
 $abcde_{(f)} = (a \cdot f^4 + b \cdot f^3 + c \cdot f^2 + d \cdot f^1 + e \cdot f^0)_{(10)}$

MSB: the sign of the binary number - 1(negative), 0(positive)

- **2's complement representation**
 - negative - 1 as the sign

$$1111_{(2)} = -1 \cdot 2^3_{(10)} + 1 \cdot 2^2_{(10)} + 1 \cdot 2^1_{(10)} + 1 \cdot 2^0_{(10)} = -1_{(10)}$$

- Variable&Constants

Identifier

- a **variable** (covered in this lesson)
- a **constant** (covered in this lesson)
- a parameter
- a user-defined method
- a user-defined class

cannot be a keyword, eg. double double = 2.3

Variable

Must be declared and initialized before using it

Can be changed

```
1 //Eg.
2 int a=b=1;
3 a=2;
4 // 最终 : a=2, b=1
```

Constants

Constant name is all **CAPITLIZED**

Use the keyword **final**

```
1 //Eg.  
2 final double PI = 3.14
```

- Expression

Types

Arithmetic Expression (with or without variables)

- double a = 2.0 * 3.0;
- int b = a + 2;

Boolean Expression (operators to be covered later)

- boolean a = true;
- boolean b = a && false;

Method Calls (details covered in later lessons)

- int sideLength = 10;
- double area = Math.pow(sideLength, 2);

Arithmetic Expression

Operation	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%

For an operation involving a double and int, int is promoted to a double

Eg. double a = 3.5 + 2;

Truncation - int/int

```
1 int a = 1/2; // -> 0
2 int a = 1/2*1; // -> 0
3 double a = 1/2; // -> 0.0
4 double a = 1/2*1.0 // -> 0.0
5 double a = 1*1.0/2 // -> 0.5
```

String Concatenation

```
System.out.println(false+"1");
System.out.println(1+"1");
System.out.println(1+1+"1");
System.out.println("1"+1+1);
System.out.print(false+1);
```

false1

11

21

111

compile error

Relational Operators

Operator	Meaning	Example
==	equal to	if (x == 100)
!=	not equal to	if (age != 21)
>	greater than	if (salary > 30000)
<	less than	if (grade < 65)
>=	greater than or equal to	if (age >= 16)
<=	less than or equal to	if (height <= 6)

- Only for primitive Types!

Eg. String(object) are compared using equals() & compareTo()

- Do not use `==` to test for equality of floating-point numbers! since they cannot always be represented exactly in the computer memory.

Eg. `4.4%3 >>> 1.4000000000000004`

Compound Assignment Operators

Operator	Example	Meaning
<code>=</code>	<code>x = 2</code>	simple assignment
<code>+=</code>	<code>x += 4</code>	<code>x = x + 4</code>
<code>-=</code>	<code>y -= 6</code>	<code>y = y - 6</code>
<code>*=</code>	<code>p *= 5</code>	<code>p = p * 5</code>
<code>/=</code>	<code>n /= 10</code>	<code>n = n / 10</code>
<code>%=</code>	<code>n %= 10</code>	<code>n = n % 10</code>

Increment and DEcrement Operators

Operator	Example	Meaning
<code>++</code>	<code>i++</code> or <code>++i</code>	<code>i</code> is incremented by 1
<code>--</code>	<code>k--</code> or <code>--k</code>	<code>k</code> is decremented by 1

Difference between `i++` and `++i`

`i++` means use the original value of `i` first before increment

`++i` means increment `i` first then use the value of the incremented `i`

Logical Operators

Operator	Meaning	Example
!	NOT	if (!found)
&&	AND	if (x < 3 && y > 4)
	OR	if (age < 2 height < 4)

PRIORITY - IMPORTANT

highest precedence	→	(1)	!, ++, --
		(2)	*, /, %
		(3)	+, -
		(4)	<, >, <=, >=
		(5)	==, !=
		(6)	&&
		(7)	
lowest precedence	→	(8)	=, +=, -=, *=, /=, %=

Short Circuit

- A&&B
 - A 为 false, B 不运行
- A||B
 - A 为 true, B 不运行

- De Morgan's Law

- **!(A&&B) = !A || !B**

- $\neg (A \vee B) = \neg A \wedge \neg B$

- Decision-Making Control Structures

If Statements

```
1 if(expression) // expression must have a boolean value
2 {
3     Statement1; //executed only when expression is true
4 }
```

If-else Statements

```
1 if(expression)
2 {
3     Statement1;
4 }
5 else if(expression)
6 {
7     Statement2;
8 }
9 else
10 {
11     Statement3;
12 }
```

if 后可以不加大括号 {}, 但只能控制 *if* 后一行的代码

```
1 int a = 0;
2 if(a==1)
3     System.out.print(1); // S1
4     System.out.print(2); // S2
5 //Output: 2
6 //因为无法进入if statement, S1无法运行;
7 //但S2不属于if statement,所以正常运行
8
9 if(a==0){
10     System.out.print(1);
11     System.out.print(2);
12 }
13 //Output: 12
```

Lesson 6-9

10-3 Dennis 袁梓晨

Lesson 6

Rewrite of nested if statement

nested if statement has several if and else-if statement that implanted in one logic

Example:

```
1  if (expr 1){
2      if(expr 2){
3          oper 1;
4      }
5      else{
6          oper 2;
7      }
8  }
9  else if (expr 3){
10     if (expr 4)
11         oper 3
12     else
13         oper 4
```

we can rewrite nested if statements using compound logic

```
1  if (expr1 && expr2){
2      oper 1;
3  }
4  else if(expr2){
5      oper 2;
6  }
7  else if (expr3 && expr4){
8      oper 3
9  }
10 else{
11     oper4
12 }
```

Practice Example 1

```
1  int a=10;
2  int b=20;
3
4  if(a==10){
```



```
5
6     if(b!=20){
7         System.out.println("GeeksforGeeks");
8     }
9
10    else{
11        System.out.println("GFG");
```

output: GFG

Notice: Pay attention to the position of the implanted statements

```
1  if (expr1){ \\ outer if statement
2      if(expr2){
3          State 1;
4      } else \\implanted in the outer if statement
5          State 2;
6  }
7 }
```

Defusion of the compound logic in if statement

- && and
- || or
- ! not

notice that :

```
1  !(c == d) is equivalent to (c != d)
2  !(c != d) is equivalent to (c == d)
3  !(c < d) is equivalent to (c >= d)
4  !(c > d) is equivalent to (c <= d)
5  !(c <= d) is equivalent to (c > d)
6  !(c >= d) is equivalent to (c < d)
```

Example: loop year

Loop year has three characteristics

- divisible by 4
- not divisible by 100
- can divisible by 400

original

```
1 boolean isLoopyear;  
2 int year = //user input;  
3 if (year%4==0)  
4     if(year%400==0)  
5         return true;  
6     else if(year%100==0)  
7         return false;  
8     else  
9         return true;
```

using compound logic

```
1 if (year%4==0&&year%100!=0 || year%400==0)  
2     return true;  
3 else  
4     return false;
```

Lesson 7

While loop

a while loop will continue running until the statement is false

```
1 // if statements just run once if the condition is true  
2 if (condition){  
3     statements;  
4 }  
5 // while statements are repeated while the condition is true  
6 while (condition){  
7     statements;  
8 }
```

-Reminders

1. while (true) or while (false) is an **infinite loop**
2. make sure the truth value of the boolean **can change** during iteration
3. If the while loop is false at first, it will never be executed
4. make sure that the variable in while loop **may be refreshed** everytime the statement executed

-Infinite loop it is not a compile error, and the program will continue executing

-while loop with if statements

Example

```
1 int i = 0;
2 int a = 0;
3 while i<10{
4     if (i%2==0){
5         a++
6     }
7 }
8 ...print(a)
```

Notice: Remember these 3 steps to writing a loop:

1. Initialize the loop variable (before the while loop)
2. Test the loop variable (in the loop header)
3. Change the loop variable (in the while loop body at the end)

Practice example 1

Consider the following code segment. Which of the following can be used as a replacement for the missing loop header so that the loop prints out “0 2 4 6 8 10”?

```
1 int count = 0;
2 /* missing loop header */
3 {
4     System.out.print(count + " ");
5     count += 2;
6 }
```

answer: while (count <= 10)

Lesson 8

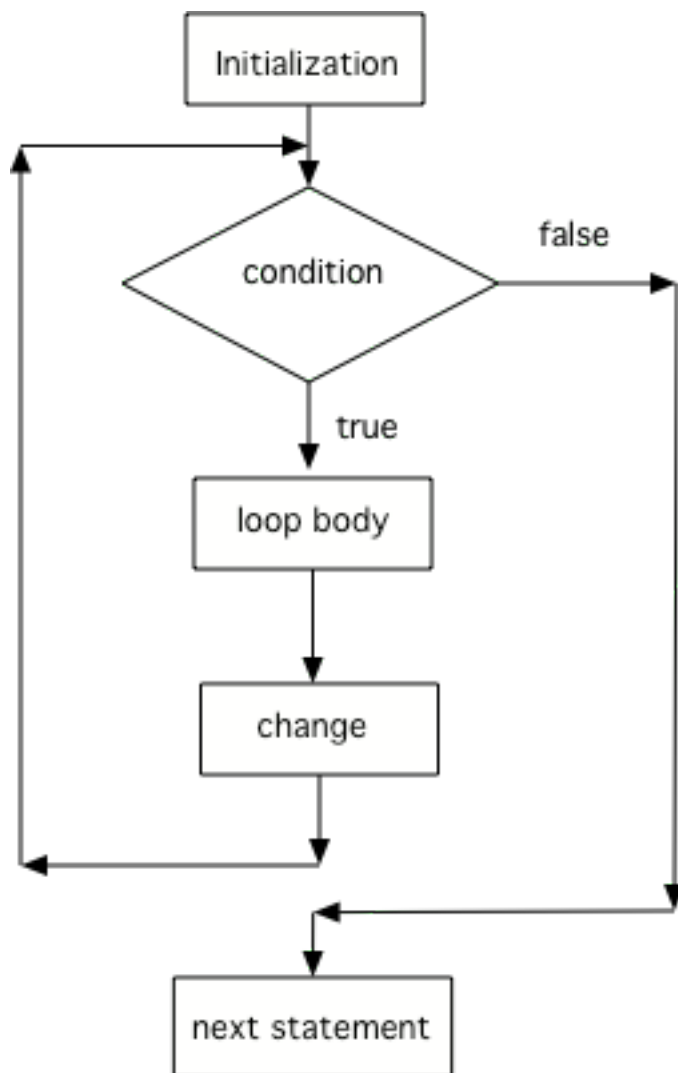
-For loop

For loops have three parts

- initialize
- test condition
- update

If test condition remains true, the loop will be executed; if false, execution will continue at the next statement after the body of the loop.

Notice that initialization and update are optional to maintain the loop

*figure 8.1***-Comparison between while & for loop**

For loop is a more clear way to show a while loop:

<pre>for (int x = 3; x > 0; x--) { System.out.println(x); }</pre>	<pre>int x = 3; while (x > 0) { System.out.println(x); x = x - 1; }</pre>
--	--

Arrows indicate the mapping between the for loop components and the while loop components: the initialization 'int x = 3;' corresponds to the first part of the for loop's parentheses; the condition 'x > 0' corresponds to the middle part of the parentheses; and the update 'x--' corresponds to the final part of the parentheses.

figure 8.2: for loop & while loop switch

We can always rewrite a while loop into a for loop, rewrite a for loop into a while loop.

-Infinite for loop

an infinite for loop occurs due to a test condition that always remain true

```
1 for (int i = 100, i>0){
2     i++
3 }
4 Infinite loop!
```

-off by one error

It is a common situation that we may get a wrong output that we want due to off by one error, that is often occur due to the wrong test condition.

Notice that in each iteration of a for loop, the increment or decrement statement is executed after the entire loop body is executed and before the Boolean expression is evaluated again.

```
1 for (int num=0; num<=10; num++)
2 output: 11
```

Practice Example 1

```
1 for (int i = 3; i <= 9; i++)
2 {
3     System.out.print("*");
4 }
```

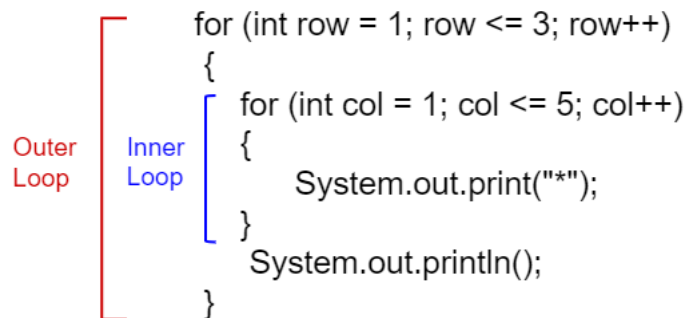
How many times does * printed? 7

Lesson 9**Nested loops**

Examples of Iterations and loops that can be write as nested loops

- while loop
- for loop
- nested control structure

A nested loop is literally a group of loops that occurs in sequential order



```
for (int row = 1; row <= 3; row++)  
{  
    for (int col = 1; col <= 5; col++)  
    {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

The diagram shows a nested for loop. A red bracket on the left side of the outer loop (the first for loop) is labeled "Outer Loop". A blue bracket on the left side of the inner loop (the second for loop) is labeled "Inner Loop".

Figure 9.1: nested for loop

Notice that in each iteration of the outer loop, the inner loop will be re-started. The inner loop must finish all of its iterations before the outer loop can continue to its next iteration.

Example of nested for loop

```
1 for (int row = 1; row <= 3; row++)  
2 {  
3     for (int col = 1; col <= 5; col++)  
4     {  
5         System.out.print("*");  
6     }  
7     System.out.println();  
8 }
```

Example of nested while loop

```
1 while(cond_outer){  
2     con1  
3     while(cond_inner){  
4         con2  
5     }  
6 }
```

Practice Example 1

Print out the following right triangle

```
1 ****  
2 ***  
3 **  
4 *
```

Program:

```
1  for (i=0, i<4, i++){
2      for (j=0, j<4-i, j++){
3          ...print(' *')
4      }
5      ...println()
6  }
```

Practice example 2

print a isoceles triangle

```
1  int row = 5;
2  int col = 2*row-1;
3  for (int i = 0; i<row; i++){
4      for (int j = 0; j<col/2-i; j++){
5          ...print(" ");
6      }
7      for (int j = 0; j<2*i+1; j++){
8          ...print(" *")
9      }
10     ...println();
11 }
```

Lesson 10-12

Justin 10-6

Lesson 10

the basic introduction of object-oriented programming

Object-oriented programming.

Model real-world entities as objects of a “genreal” class

a class can be thought of as a document that describes the properties or functionalities objects/instances of that class

what are the components of a class

1. class header

2. instance and static variables- represent properties.
3. constructors
4. methods

writing class: public, capital the first letter in the name of the class. for example

```
1 public class BankAccount{  
2 }
```

Instance variables

represent properties any of instance of the class.

Instance must be declared as private.

Not initialized, not final

Examples

```
1 public class BankAccount{  
2     //instance variables:  
3     private String password;  
4     private double balance;  
5 }
```

Static variables:

represent properties to the entire class

have the same value for all instances of the class

```
1 Public class BankAccount{  
2     //static variables  
3     public static final double overdrawn-limit=0.0;  
4 }
```

public and private modifiers

private: visible within the scope of class

public: can be access with outside the class

constructor

initialize instance variables when object is being created during runtime

```
1 public class BankAccount{
2     // instance variable
3     private String password;
4     private double balance;
5
6     //zero-parameter constructor
7     public BankAccount(){
8         //the same with password="", balance=0.0;
9         public BankAccount(String password, double balance)
10        {
11            this.password=password;
12            this.balance=balance;
13        }
14        //this. refers to the line 3 and line 4, it means this variable is
15        //the class instance objects
16    }
17 }
```

Syntax: public className(...){...}

When no constructor is written, Java provides a default zero-parameter

This.

This. is use for the class instance variable. Its use is to avoid the same name with the local variable.(However, you can still use another name for local variable)

Create object in runtime

put the main method in a runner class

create object using the new keyword followed by the constructor call

Syntax: className variableName=new className(parameters)

example:

```
1 public class BankAccountRunner{
2     public static void main(String[] args)
3     {
4         //Create a BankAccount object(call constructor)
5         BankAccount a = new BankAccount("123456",0);
6         BankAccount b = new BankAccount();
7         //use difference constructors
8     }
9 }
```

```
8     }  
9 }
```

a and b are object reference variables, which are storing address of the actual objects

Methods

Accessor(get) method

mutator(set) method

Syntax

```
public returnType methodName(type param1.....){  
    ....  
}
```

When returnType is not void, there must be a return statement, which returns the value of the type

When it is void, there is no return statement

examples of accessor and mutator

```
1 public class BankAccount{  
2     //accessor  
3     public double getBalance(){  
4         return balance;  
5     }  
6     //mutator  
7     public void deposit(double amount){  
8         balance += amount;  
9     }  
10 }
```

encapsulation

the property of the object is private, so there is an implementation: private instance variables + public methods. It can take the instance objects outside of the class

```
1 public class BankAccount{  
2     private double balance;  
3     public double getBalance(){  
4         return balance;  
5     }  
6     //since balance is private, we cannot use it outside the class. The  
    method helps us to get the value of balance
```

```
7 }
```

invoking instance methods in client class

syntax: `variableName.methodName(param1...)`

variable name must have already stored the address of an object.

static method:

static methods are methods belonging to the class

Instance variables are not allowed to be accessed

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

Lesson 11

instance method invocation

call the method directly within other instance methods in the same class

equivalent to using this.

example

```
1 public class BankAccount{
2     public double getBalance(){
3         return balance;
4     }
5     public double deposit(double amount)
6     {
7         double newAmount=getBalance()+amount;
8         return newAmount;
9     }
10    //remember the parentheses
11 }
```

outside: use `objectReference. MethodName(...)`

Static method invocation

use the static method directly inside all method

equivalent to using `className.methodName(...)`

use `className.methodName`

for example

```
1 public class BankAccount{
2     public static void printlimit()
3     {
4         System.out.print(Overdeawn-LIMIT)
5     }
6 }
7 public class BankAccountrunner{
8     BankAccount,printlimit;
9     BackAccount a=new BankAccount();
10    a.printlimit();
11 }
```

Other examples: Math classes

a^b : `Math.pow(a,b)`

the square root of a: `Math.sqrt(a)`

absolute value of a: `Math.abs(a)`

`Math.random()`: give a random number bewteen 0 and 1(not containing 1)

examples:

find a random integer 0 and 100

`int result=(int)(101*Math.random())`. Because it does not include 1, so we need to use 101

toString

when an object reference variable is printed

If doesn't store anything: `NullPointerException`

When the reference variables stores the address of the object: `toString()` method is invoked, which returns a `String`.

example

```
1 public class BankAccount{
2     public String toString()
3     {
4         return "A";
5     }
6 }
```

```
7 public class Runner{
8     BankAccount bob;
9     bob=new BankAccount(...);
10    System.out.print(bob);
11    //the same with bob.toString()
12 }
```

Parameter passing during the method invocation

we can pass primitive type or variables storing a value

object reference type

use a copy of the address

Method overload

more than one method with the same names, but different parameters.

- same method name, different number of parameters.
- different number of parameters
- different types of parameters
- parameters listed in different orders
- be careful to distinguish the parameters type

Wrapper classes

Classes/objects corresponds to primitive types.

Convenient

Convert primitive type variable initialization to object type variable initialization

example

```
1 int a1=40;
2
3 Integer a2= new Integer(40); capitalize
4
5 double b1=3.14;
6
```

```
7 Double b2= new Double(3.14);  
8  
9 System.out.print(a2.intValue());  
10  
11 System.out.print(b2.doubleValue());
```

Autoboxing

allowed to directly assign a primitive type value to the corresponding wrapper class.

```
1 int a=1;  
2 Integer x=a; // (equals to new Integer(1))
```

Double x=(int value) not allowed. It will not do autoboxing and change of the double and int value at the same time

Autoboxing:

```
1 Integer x=new Integer(2);
```

assign a object reference to its corresponding primitive type variable.

```
1 int a=x;  
2 a=2
```

Wrapper classes are immutable

```
1 Integer a1=1;  
2 a1=2;
```

there are two objects

Lesson 12

String is an object type.

Syntax: variableName=new String(String literal);

variable Name=String literal;

The first one creates a new object. The second one means it is in the String Constant pool

String initialization 1

zero parameter: `new String()`

one-parameter: `new String("")`

If you use this type of initialization, it creates new objects.

```
1 String a=new String("c");
2 String b=new String("c");
```

There are two objects.

String initialization 2

```
1 String a="a";
2 String b="a";
```

This will only create one object because they both refer to "a" in the String constant pool

"==" operator

`==` compares the contents.

```
1 String a="c";
2 System.out.print(a=="c");
3 Output: true
```

the content of a and the address of "c"

Another example:

```
1 String a=new String("c");
2 String b=new String("c");
3 System.out.print(a==b);
```

They have different addresses, so it will output: false.

Equals method(boolean)

```
1 String a=new String("c");
2 String b=new String("c");
3 System.out.print(a.equals(b));
```

Compare the values, which is the same.

Length method(int)

Calculate the length of the string

substring method(string)

one parameter: (int begin index)

remember that the first index is 0. The substring will be the part from the begin index to the end.

two parameter: (int begin index, int end index)

The substring will be the part from the begin index to the end index.

indexOf method(int)

find out the first substring which appears in the string. It will return the first letter's index

Compareto(string)

1. if two string are identical, return 0
2. if the first string is longer than the second one, and the content of the first few letters of the first string and the second one is the same, it will return how many units A more than B
3. If there is different letters, just make a calculations according to the ASCII table

Appendix

Resources

 Click to visit website

Online Java IDEs

- riju (recommended)
- judge0
- replit

Offline Java IDEs

- IDEA (recommended)
- VS Code (external plugin required)

Java resources

- Java Cheatsheet
- Java Reference

Credits

- Ana Jiang 10(3) : Management, Lesson 1~5
- Dennis Yuan 10(3) : Lesson 6-9
- Jett Chen 10(8) : Formatting
- Justin 10(6) : Lesson 10-12