**OOP Lab 6**

|  |  |
| --- | --- |
| Due Date: | **October 11, 23 : 59** |

* **Submit your assignment using the following file format:**

LabNumber\_RoomNumber\_StudentName\_Student\_ID.zip

Example: Lab6\_328\_Hongkildong\_201620505.zip

* This zip file will contain **two types of** files, namely:

1. **report file** with file format **“Report\_Lab number**” (eg. report\_6) to answer theory questions and to write the screen shot of your program.
2. Source code file that contains codes of classes to answer programming questions.

**Contents**

1. **In java, Objec**t class is the root class in class hierarchy. The Object class defines the following common methods shared by any class.
2. public **String** **toString**()
3. public native int **hashCod**e()
4. public boolean **equals(Object** o)
5. public final Class **getClass**()
6. **This lab also cover : Class** class, **String** class, **ArrayList** class and **Integer** class

**Problem 1: Use of “Class” class and reflection API. The following code has an error.**

L1. **// ObjectTest.java**

L2: public c**lass ObjectTest**

L3: {

L4: public static void **main** (String []args )**throws Exception**

L5: {

L6: int counter=0;

**L7: Class** c = Class.**forName** (“**java.lang.Object**”);

L8: **Method**[]m = c.getDeclaredMethods();

L9: **for**( **Method** mi: m )

L10: {

L11: System.out.println (mi.getName());

L12: counter++;

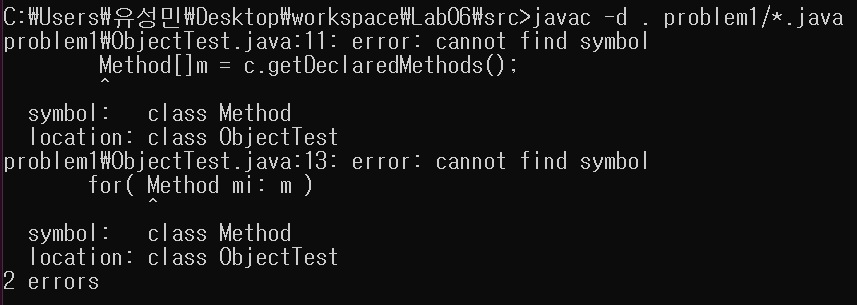
L13. }

L14: System.out.println (“the number of methods:” + count);

L15: **}// end of main**

L16: }// end of class

**1(a): What is the cause of the error? (3pt)**

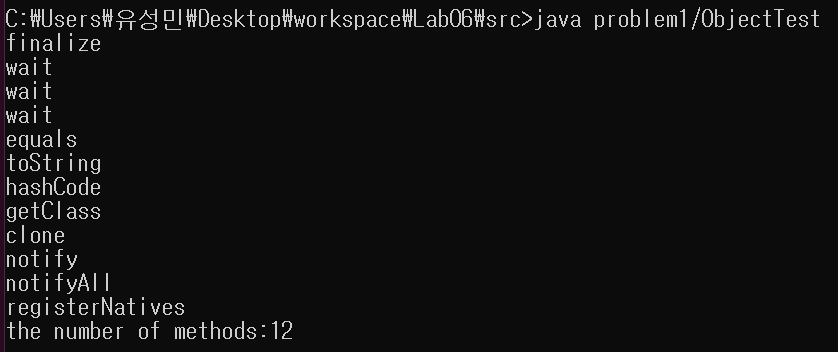
****

**<problem1 (a) error>**

**=>**

**Because the ‘class type method’ is in the “java.lang.reflect.\*”, so we should import that.**

**1(b): correct the program and show the screenshot of the program (3pt)**

****

**<problem1 (b) correct result>**

**=>**

**After add import “java.lang.reflect” in source code, then program runs well.**

**Problem 2:** **toString** () method of “**Object**” class ()

**2(a): run the following code and take the screenshot (1pt)**

**// Student.java**

L1: class **Student**

L2: {

L3: String name;

L4: int ID;

L5: public **Student**(String name, int ID)

L6: {

L7: this.name=name;

L8: this.ID=ID;

L9: }

**L10: public String toString()**

**L11: {**

**L12: return “Student name:” + name + ”and ID:” + ID;**

**L13: }**

L14: **public** static void main (String []args )

L15: {

L16: Student S1 = new Student(“kim”, 101);

L17: Student S2 = new Student (“Homin”, 102);

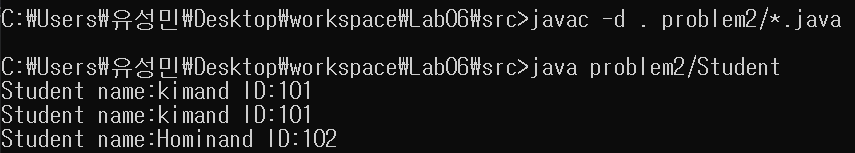
L18: System.out.println (S1); **/\* 1 \*/**

L19: System.out.println (S1.toString()); /\* **2** \*/

L20: System.out.println (S2); /\* **3** \*/

L21: }

L22:}

****

**<problem2 (a) result>**

**=>**

**The result shows the overridden toString() method.**

**So we get the results like above screen shot.**

**2(b): From 2(a), comment Line 10-13 and run the program. Show the screen shoot. Why the screen shot is different from 2(a)? Explain your answer. Hint: the** standard implementation of **toString()** method in the **Object** class is as follows (3pt).

**L1: public String toString()**

**L2: {**

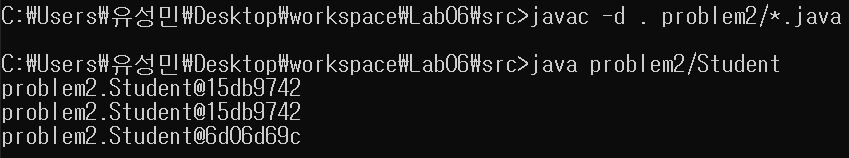
**L3: return getclass().getName()+ ”@” +Integer.toHexString(hashCode());**

**L4: }**

// a) getclass(): method of “Object” class

// b)getName(): method of “Class” class

// c)Integer.toHexString(): method of Integer wrapper class

 // d)hashCode(): method of the “**Object**” Class

**<problem2 (b)result>**

**=>**

**First, we override the toString method to return Student name and ID, but by commenting the line 10 ~ 13, overridden toString method is ignored. So the toString method at “Object class” is invoked.**

**As you gave us the standard implementation of toString() method in “Object class”,**

**the method return getclass().getName() + “@”+Integer.toHexString(hashCode());.**

**So we get the result like above screen shot.**

**Problem 3: toString()and hashCode() methods of Object class. Answer all the questions based on the following code.**

**// Test.java**

L0: import java.util.\*;

L1: **public class Test**

L2: {

**L3:**

**L4: public** static void main (String []args )

L5: {

L6: **String** **s** =new String (“kim”);// built-in **String** class

L7: System.out.println (**s**);

L8: **Integer** **I** =new Integer (10);// built-in **wrapper** class

L9: System.out.println (**I**);

L10 **ArrayList** L =new **ArrayList** (); // built-in **collection** Class

L11: L.**add**(“A”);

L12 L.**add**(“B”);

L13: System.out.println (**L**);

L14: **Test** **t** =new Test();// **user defined class**

L15: System.out.println (**t**);

L16: }

L17 }

3(a): Is java.lang.**String** class **override** **toString**() method of “Object” class? Explain your reason. Hint: refer line 7 (**2pt)**

**=>**

**Of course yes. Because we override the string to KIM. At java API, you can see that toString overrides toString in class Object, and returns the string itself.**

3(b): Is java.lang.**Integer** class **override** **toString**() method of “Object” class? Explain your reason. Hint: refer line 9(**2pt**)

**=>**

**Of course yes. At java API, it says that “toString() of Integer returns a String object representing this Integer’s value. And the value is converted to signed decimal representation and returned as a string.” So the result shows the Integer’s value 10.**

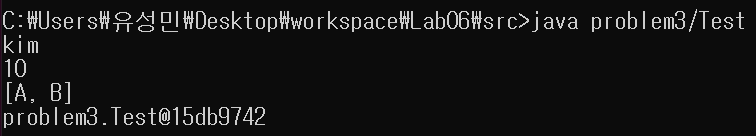
3(c): Is java.util.**ArrayList** class **override** **toStrin**g() method of “Object” class? Explain your reason. Hint: refer line 13(**2pt)**

**=>**

**No. At java API, it says that “toString() returns a string representation of this collection. The string representation consists of a list of the collection’s elements in the order they are returned by iterator, enclosed in square bracket(“[]”). Adjacent elements are separated by the characters.” So we just used the toString() without overriding.**

3(d): Is Test class **override** **toString**() method of “**Object**” class? Explain your reason. Hint: refer line 14(**2pt**)

**=>**

**No. Because there is no relationship between other classes’s(String class and Integer class) toString() method overriding and Test class’s toString() method.**

**<problem3 (a),(b),(c),(d) result>**

**Program 4: Overriding toString()method by Overriding only hashcode()method. Answer all the question based on the following code.**

**// Test.java**

L1: **class Test**

L2: {

L3: **int i;**

L4:

L5: public **Test** (int i);

L6: {

L7: this.i=i;

L8: }

**L9**: **public int hashCode().**

**L10: {**

**L11: return i;**

**L12: }**

**L13: public String toString().**

**L14: {**

**L15: return** **i** **+** ” “;

**L16: }**

L17: public **static** void main (String []args)

L18: {

L19: Test t1 = new Test (10);

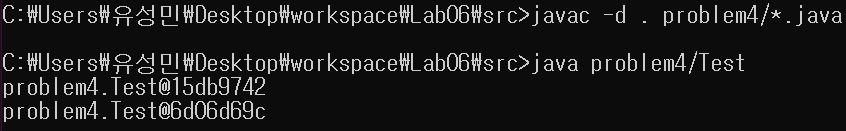
L20: Test t2 = new Test (100);

**L21: System.out.println (t1);**

**L22: System.out.println (t2);**

L23: }

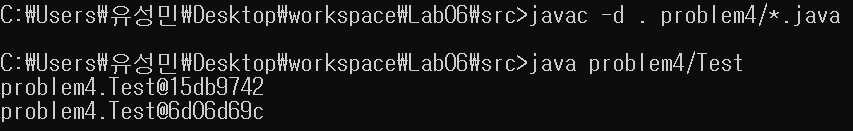
L24: }

****4(a) comment Line **9-16** and see the output. Why you get this output?(**3pt**)

**<problem4 (a) result>**

**=>**

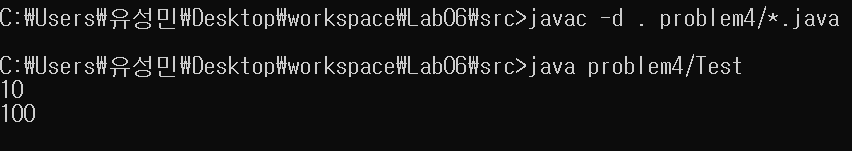
**First at line 9~16, we override the hashcode method to return ‘int I’ So when the code execute System.*out*.println(t1); and System.*out*.println(t2); two overridden method(hashcode and toString) of Test class is invoked and the results show 10 and 100. But after commenting the line, overridden methods of Test class are deleted so two methods of “Object” class are invoked**.

4(b) comment Line **13-16** and see the output. Why you get this output?(**3p**t)

**<problem4 (b) result that is same above result>**

**=>**

**At line 13~16, we only override hashcode method but not toString method. By commenting the toString method of Test class, the toString method of Object class is invoked and the toString method of Object class invokes hashcode method. But we override the hashcode method to return ‘int I’ instead of hashcode with hex numbers. So the results show 10 to ‘a’ in hex and 100 to 64 in hex.**

4(c) remove the comment **9-12** and see the result. Why you get this output?(**3pt**)

**<problem4 (c) result>**

**=>**

**At line 9~12, we only override toString method but not hashcode method. By commenting the hashcode method of Test class, As a result, the hashcode method is ignored and only the overridden toString function is executed, resulting in the values ​​10 and 100 that we put in the constructor.**

**Problem 5:** Difference between operator (= =) and **equals ()** method in “**Object**” Class

Remark 1: Assume that the meaning of equality is to comparing only names, only roll numbers or both.

Remark 2: When we pass heterogeneous objects, avoid rising of Class Cast exception

Remark 3: When we pass null argument, avoid rising of Null Pointer Exception

**5(a). Run the following Code and take screen shoot (1pt).**

**// Student.java**

L1: class **Student**

L2: {

L3: **String** name;

L4: **int** ID;

L5: public **Student** (String name, int ID)

L6: {

L7: this.name=name;

L8: this.ID=ID;

L9: }

**L10: public boolean equals(Object obj) // S1.equal(S2)**

**L11: {**

**L12: try**

**L13: {**

**L14 String name1 =this.name; // this refers to S1.**

**L15: int ID1=this.ID;**

**L16; Student s =(Student)Obj;**

**L17: String name2 =s.name;**

**L18: int ID2 =S.ID;**

**L19: if( name1.equals(name2)&& (ID1==ID2))**

**L20: return true**

**L21: else**

**L22: return false;**

**L23: }// end of try block**

**L24: catch(** **ClassCastException e)**

**L25: {**

**L26:** return false;

**L27: }**

**L28 catch(** NullPointer**Exception e)**

**L29: {**

**L30:** return false;

**L31: }**

**L32: } // end of equals()**

L33: **public** static void main (String []args )

L34: {

L35: **Student S1 = new Student(“kim”, 101);**

L36: Student S2 = new Student (“Homin”, 102);

L37: **Student S3 = new Student (“Homin”, 102);**

L38: **Student S4 = S1**

L39: System.out.println ( **S1.equals(S2**));

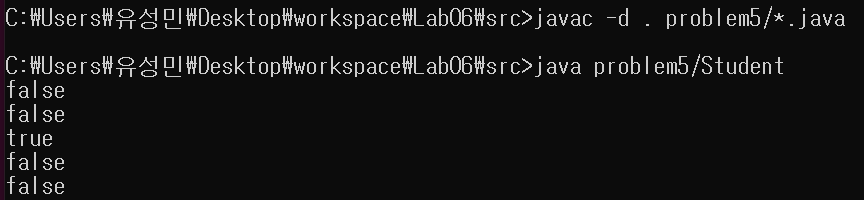
L40: **System.out.println ( S1.equals(S3));**

L41: System.out.println ( **S1.equals(S4**));

**L42: System.out.println ( S1.equals(“kim”));**

**L43 System.out.println ( S1.equals(null));**

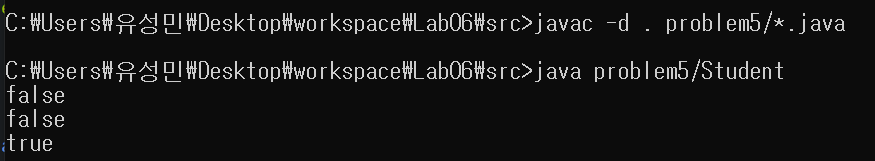
**L44: } end of main**

**L45: }// end of class**

**<problem5 (a)>**

**b) In the above code, comment Line 10-32 and comment lines 42-43.**

**What is the output at line 39, 40 and 41? Why you get this output (3pt).**

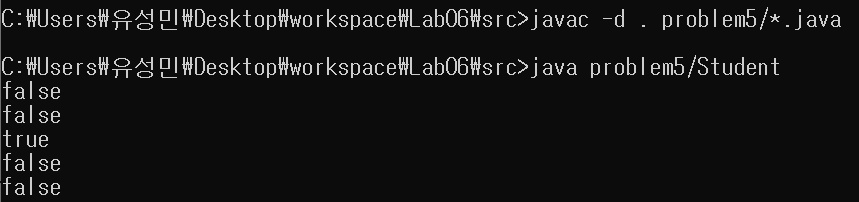
****

**<problem5 (b) result>**

**=>**

**line 10-32: We override the equals() method and this method compare the fields(variables) of the object. Plus, as you can see the line 24-31, we also handle the exceptions.**

**line 42-43: line 42 compare S1’s variables and “KIM”, but we should compare not only name but also ID. So the result shows false. line 43 compare S1 and null, but as we can see Student S1 = new Student(“KIM”, 101);, S1’s variables were initialized to KIM and 101. So the result shows false.**

**c) In the above code, comment only Line 10-32. Is there error at line 42 and Line 43? Why? (3pt).**

**<problem5 (c) result>**

**=>**

**NO. There is no error. The Student class overrides equals() method and its function is to compare the members which object has inside. So when equals method of Student class was invoked, it compares the members of S1 and S2, S3, S4, KIM, null. Only S4 was same as S1, so the result was true(other comparing results in false). But after comment the line 10-32, equals() method of “Object” class was invoked which compares references of object. Those S1, S2, S3 are different instances and S1 was assigned to S4, so S1 is same with only S4. S1’s reference is different with S2, S3, KIM, null, so the results of comparing with those were false. So the result is same whether comment the line or not. As a result, result of using equals() of “Object” class and “Student” class are same.**

**Problem 6: Replace the line 10-32 in Problem 5(a) by the following code. Did you get the same result as in 5(a) in the problem 5? Explain your reason (3pt)**

**L0: Public oolean equals(Object obj)// S1.equal(S2)**

**L1: {**

**L2: try**

**L3: {**

**L4; Student s =(student)Obj;**

**L5: if( name.equals(s.name)&& (ID==S.ID) )**

**L6: return true**

**L7: else**

**L8: return false;**

**L9: }// end of try block**

**L10: Catch(** **ClassCastException e)**

**L11: {**

**L12:** return false;

**L13: }**

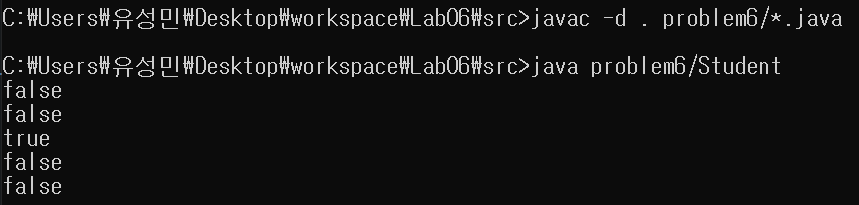
**L14 Catch(** NullPointer**Exception e)**

**L15: {**

**L16:** return false

**L17: }**

**L18: }// end of equals()**

****

**<problem6 result>**

**=>**

**Yes. This overridden equals() method compares the object which is given to parameter and also compares members that object has. So the result is same as task 1 in problem 5(a).**

**Problem 7: Replace the line 10-32 in Problem 5(a) by the following code. Did you get the same result as in (5a) in problem 5? Explain your reason (4pt)**

**L1: public boolean equals(Object obj) // S1.equal(S2)**

**L2: {**

**L3; if(obj instanceof Student)**

**L4: {**

**L5: Student s =(student)Obj;**

**L6: if( name.equals(s.name)&& (ID==S.ID) )**

**L7: {**

**L8: return true**

**L9: }**

**L10: else**

**L11 {**

**L12: return false;**

**L13: }**

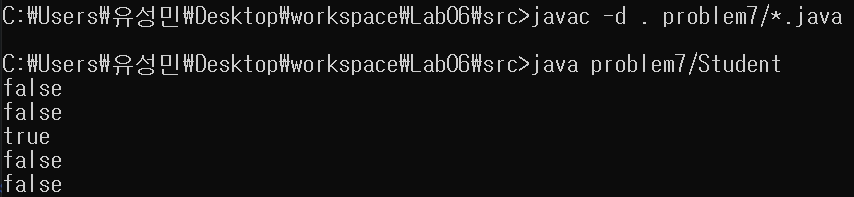
**L14: }// end of if block**

**L15: else**

**L16: {**

**L17 return false;**

**L18: }**

** L19: }**  **end of equals()**

**<problem7 result>**

**=>**

**Yes it shows same result with problem 5. This overridden equals() method firstly distinguish whether the object which is given by parameter is instance of the Student object. If not it returns false and if the object is instance of the Student object then compares the members of the two references. So the result shows same as with problem 5.**

**Problem 8: Replace the line 10-32 in Problem 5 by the following code. Did you get the same result as in 5(a) in problem 5? Explain your reason (4pt)**

**L1: public boolean equals(Object obj) // S1.equal(S2)**

**L2: {**

**L3: if(obj == this)// instead of comparing field by field**

**L4: {**

**L5: return true;**

**L6: }**

**L7; if(obj instanceof Student)**

**L8: {**

**L9: Student s =(student)Obj;**

**L10: if( name.equals(s.name)&& (ID==S.ID) )**

**L11: {**

**L12: return true**

**L13: }**

**L14: else**

**L15 {**

**L16: return false;**

**L17: }**

**L18: }// end of if block**

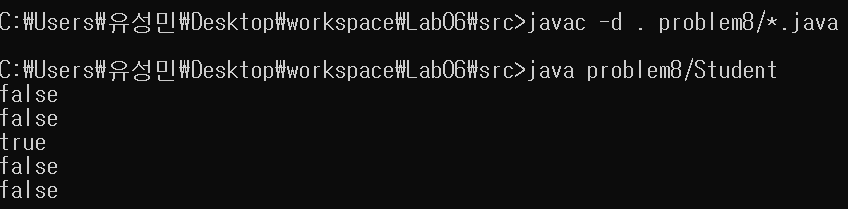
**L19: else**

**L20: {**

**L21 return false;**

**L22: }**

**L23: }**  **end of equals()**

****

**<problem8 result>**

**=>**

**Yes. The result is also same as the one of problem 5. This equals() method compares references first, and if two references are not same then compares the members of two references.**

**So reference s1 is firstly compared with those references(s2, s3, s4, KIM, null) and then compares s1’s members with other references members. As you can see, s1’s reference is only same as s4’s, and s1’s member is only same as s4’s. So result is same with result of problem 5.**