Object-Oriented Programming with Java

Recitation No. 13

String Interning

```
public static void main(String args[]) {
    if ("hello".substring(0) == "hello")
        System.out.println("statement 1 is true");

    if ("hello".substring(1) == "ello")
        System.out.println("statement 2 is true");

    if ("hello".replace('l','l') == "hello")
        System.out.println("statement 3 is true");

    if ("hello".replace('h','H') == "hello".replace('h','H'))
        System.out.println("statement 4 is true");

    if ("hello".replace('h','H') == "Hello")
        System.out.println("statement 5 is true");
}
```

The output is:

```
statement 1 is true
statement 3 is true
```

String Interning

```
public static void main(String args[]) {
    String s1 = "he";
    String s2 = "he" + "llo";
    String s3 = s1 + "llo";

    if (s2.equals(s3))
        System.out.println("statement 1 is true");
    if (s2 == "hello")
        System.out.println("statement 2 is true");
    if (s2 == s3)
        System.out.println("statement 3 is true");
    if (s2 == new String("hello"))
        System.out.println("statement 4 is true");
    if (s2 == s3.intern())
        System.out.println("statement 5 is true");
    if (s3 == s2.intern())
        System.out.println("statement 6 is true");
}
```

The output is:

```
statement 1 is true
statement 2 is true
statement 5 is true
```

Visibility

```
public class A {
    private int bar =0;

    public boolean isEqual(A a) {
        return (bar == a.bar);
    }

    public static void main(String[] args) {
        A a1 = new A();
        A a2 = new A();
        System.out.println(a1.isEqual(a2));
    }
}
```

The output is:

```
true
```

No compilation error:

Objects of the same class can access each other's private fields

Inheritance

Consider the following class hierarchy:

Interface **Animal** {…}
class **Dog** implements Animal{…}
class **Poodle** extends Dog {…}
class **Labrador** extends Dog {…}

Which of the following lines (if any) will not compile?

Poodle poodle = new Poodle();
Animal animal = (Animal) poodle;
Dog dog = new Labrador();
animal = dog
poodle = dog;

Poodle = (Poodle) dog;
-No compilation error
-Runtime Exception

Method Overriding

```
public class A {
   public void print() {
      System.out.println("A");
     }
}

public class B extends A {
   public void print() {
      System.out.println("B");
    }
}
```

```
public class C {
     public static void main(String args[]) {
            B b = new B();
            A a = b;

            b.print();
            a.print();
     }
}
```

Casting is unneeded

The output is:

```
B
B
```

Method Overriding & Visibility

```
public class A {
    public void print() {
        System.out.println("A");
    }
}

public class B extends A {
    protected void print() {
        System.out.println("B");
    }
}
```

```
public class C {
    public static void main(String[] args) {
        B b = new B();
        b.print();
    }
}
```

Compilation error:
"Cannot reduce the visibility of the inherited method"

Method Overriding & Visibility

```
public class A {
    protected void print() {
        System.out.println("A");
    }
}

public class B extends A {
    public void print() {
        System.out.println("B");
    }
}
```

```
public class C {
    public static void main(String[] args) {
        B b = new B();
        b.print();
    }
}
```

The output is:

```
B
```

Inheritance & Constructors

```
class A {
      B b = new B();
      public A() { System.out.println("in A: no args."); }
      public A(String s) { System.out.println("in A: s = " + s); }
}

class B {
      public B() { System.out.println("in B: no args."); }
}

class C extends A {
      B b;
      public C() { System.out.println("in C: no args."); }
      public C(String s) { System.out.println("in C: s = " + s); }
}

Class D {
      public static void main(String args[]) {
            C c = new C();
            A a = new C();
      }
}
```

The output is:

```
in B: no args.
in A: no args.
in C: no args.
in B: no args.
in A: no args.
in C: no args.
```

Inheritance & Constructors

```
class A {
    B b = new B();
    public A() { System.out.println("in A: no args."); }
    public A(String s) { System.out.println("in A: s = " + s); }
}

class B {
    public B() { System.out.println("in B: no args."); }
}

class C extends A {
    B b;
    public C() { System.out.println("in C: no args."); }
    public C(String s) { System.out.println("in C: s = " + s); }
}

class D {
    public static void main(String args[]) {
        C c = new C("c");
        A a = new C("a");
    }
}
```

The output is:

```
in B: no args.
in A: no args.
in C: s = c
in B: no args.
in A: no args.
in C: s = a
```

Inheritance

```
public class A {
    String bar = "A.bar";

    A() { foo(); }

    public void foo() {
        System.out.println("A.foo(): bar = " + bar);
    }
}

public class B extends A {
    String bar = "B.bar";

    B() { foo(); }

    public void foo() {
        System.out.println("B.foo(): bar = " + bar);
    }
}
```

```
public class D {
    public static void main(String[] args) {
        A a = new B();
        System.out.println(a.bar);
        a.foo();
    }
}
```

The output is:

```
B.foo(): bar = null
B.foo(): bar = B.bar
A.bar
B.foo(): bar = B.bar
```

Inheritance

```
public class A {
	String bar = "A.bar";
}

public class B extends A {
	String bar = "B.bar";
	B() { foo(); }
	public void foo() {
		System.out.println("B.foo(): bar = " + bar);
	}
}
```

```
public class D {
	public static void main(String[] args) {
		A a = new B();
		System.out.println(a.bar);
		a.foo();
	}
}
```

Compilation Error:
"The method foo is undefined for the type A"

Inheritance

```
public class A {
    private void foo() {
        System.out.println("A.foo()");
    }

    public void bar() {
        System.out.println("A.bar()");
        foo();
    }
}

public class B extends A {
    public void foo() {
        System.out.println("B.foo()");
    }
}
```

```
public class D {
    public static void main(String[] args) {
        A a = new B();
        a.bar();
    }
}
```

The output is:

```
A.bar()
A.foo()
```

Inheritance

```
public class A {
        public void foo() {…}
}

public class B extends A {
    public void foo() {…}
}
```

How can you invoke the foo method of A within B?
<u>Answer</u>:
Use super.foo()

Can we use ((A) this).foo() ?
<u>Answer</u>:
No.
StackOverflowError will be thrown.

Inheritance

```
public class A {
        public void foo() {…}
}

public class B extends A {
     public void foo() {…}
}

public  class C extends B {
      public void foo() {…}
}
```

How can you invoke the foo method of A within C?
Answer:
Not possible
(super.super.foo() is illegal)

Inner Classes

```
public class Test {
    public int a = 0;
    private int b = 1;

    public void foo(final int c) {
        int d = 2;

        class InnerTest {
            private void bar(int e) {

            }
        }
    }
}
```

Only a,b,c and e are accessible at the highlighted line.

Inheritance

```
class A {
    void print() {
        System.out.println("A");
    }
}

class B extends A implements C {
}

interface C {
    void print();
}
```

Compilation error:
The inherited method A.print() cannot hide the public abstract method in C

Inheritance

```
class A {
    public void print() {
        System.out.println("A");
    }
}

class B extends A implements C {
}

interface C {
    void print();
}
```

No compilation errors

Method Overloading

```
public class A {
      public void foo(Object o) {
            System.out.println("Object");
      }

      public void foo(String s) {
            System.out.println("String");
      }

       public static void main(String args[]) {
             A a = new A();
             a.foo(null);
       }
}
```

- Does the code compile? If no, why?
- Does the code throw a runtime exception? If yes, why? If no, what is the output?

Answer: The code compiles and runs, printing "String"

Method Overloading

```
public class A {
      public void foo(StringBuffer sb) {
            System.out.println("StringBuffer");
      }

      public void foo(String s) {
          System.out.println("String");
       }

       public static void main(String args[]) {
            A a = new A();
            a.foo(null);
       }
}
```

•Does the code compile? If no, why?
•Does the code throw a runtime exception? If yes, why? If no, what is the output?

Answer: The code does not compile (an ambiguous method)

Method Overloading

```
public class A {
    private static class B {}
    private static class C extends B {}

    public void foo(B b) {
            System.out.println("B");
    }

    public void foo(C c) {
        System.out.println("C");
    }

     public static void main(String args[]) {
         A a = new A();
         a.foo(null);
    }
}
```

- Does the code compile? If no, why?
- Does the code throw a runtime exception? If yes, why? If no, what is the output?

Answer: The code compiles and runs, printing “C”

Good-Luck!!!