EECS2030 (Sec. E & F) Fall 2024
Advanced Object Oriented Programming
Example Exam Questions

Caveat: These questions are **just examples** and **not** meant to be complete.
You should prioritize your time in studying all the covered materials.

1. Assume that a `Person` class is already defined, and it has an attribute `name` and a constructor that initializes the person's name from the input string. Consider the following fragment of Java code (inside some `main` method):

```
1 Person p1 = new Person("Heeyeon");
2 Person p2 = new Person("Jiyoon");
3 System.out.println(p1 != p2);
```

   What happens when executing the above Java code?

2. Assume that a `Person` class is already defined, and it has an attribute `name` and a constructor that initializes the person's name from the input string. Consider the following fragment of Java code (inside some `main` method):

```
1 Person p1 = new Person("Heeyeon");
2 Person p2 = new Person("Jiyoon");
3 Person[] persons = new Person[2];
4 System.out.println(persons[persons.length()] != null);
```

   What happens when executing the above Java code?

3. Assume that a `Person` class is already defined, and it has an attribute `name` and a constructor that initializes the person's name from the input string. Consider the following fragment of Java code (inside some `main` method):

```
1 Person p1 = new Person("Heeyeon");
2 Person p2 = new Person("Jiyoon");
3 Person[] persons = new Person[2];
4 System.out.println(persons[persons.length] != null);
```

   What happens when executing the above Java code?

4. Assume that a `Person` class is already defined, and it has an attribute `name` and a constructor that initializes the person's name from the input string. Consider the following fragment of Java code (inside some `main` method):

```
1 Person p1 = new Person("Heeyeon");
2 Person p2 = new Person("Jiyoon");
3 Person[] persons = new Person[2];
4 System.out.println(persons[persons.length - 1] != null);
```

What happens when executing the above Java code?

5. Assume that a `Person` class is already defined, and it has an attribute `name` and a constructor that initializes the person's name from the input string. Consider the following fragment of Java code (inside some `main` method):

```
1 Person p1 = new Person("Heeyeon");
2 Person p2 = new Person("Jiyoon");
3 Person[] persons = new Person[2];
4 System.out.println(persons[persons.length - 1].name.equals("Jiyoon"));
```

What happens when executing the above Java code?

6. Assume that a `Person` class is already defined, and it has an attribute `name` and a constructor that initializes the person's name from the input string. Consider the following fragment of Java code (inside some `main` method):

```
1 Person p1 = new Person("Heeyeon");
2 Person p2 = new Person("Jiyoon");
3 Person[] persons = {p1, p2};
4 p1 = p2;
5 System.out.println(persons[0] == p1);
```

What happens when executing the above Java code?

7. Assume that a `Person` class is already defined, and it has an attribute `name` and a constructor that initializes the person's name from the input string. Consider the following fragment of Java code (inside some `main` method):

```
1 Person p1 = new Person("Heeyeon");
2 Person p2 = new Person("Jiyoon");
3 Person[] persons = {p1, p2};
4 p1 = p2;
5 persons[0] = p2;
6 System.out.println(persons[0] == p1);
```

What happens when executing the above Java code?

8. Assume that a `Person` class is already defined, and it has an attribute `name`, a constructor that initializes the person's name from the input string, and a mutator method `setName` that changes the person's name from the input string. Consider the following fragment of Java code (inside some `main` method):

```
1 Person p1 = new Person("Heeyeon");
2 Person p2 = new Person("Jiyoon");
3 Person[] persons = {p1, p2};
4 p1 = persons[1];
5 persons[0] = p2;
6 p2.setName("Jihye");
7 System.out.println(p1.name);
```

What happens when executing the above Java code?

9. Consider the following classes, where we use `print` to abbreviate `System.out.println`:

```
class A extends B {
  A() { }
}
```

```
class B extends C {
  B() { }
}
```

```
class C {
  C() { }
  void bm(){print("C.bm");}
}
```

```
class D extends C {
  D() { }
  void cm(){print("D.cm");}
}
```

```
class F extends D {
  F() { }
  void bm(){print("F.bm");}
  void em(){print("F.em");}
}
```

```
class E extends F {
  E() { }
  void dm(){print("E.dm");}
}
```

Now consider the following code in the `main` method of a tester class for the above classes:

```
1  D d1 = new C();
2  C d2 = new D();
3  d2.bm();
4  D e1 = new E();
5  d2 = e1;
6  d2.bm();
7  F f = e1;
8  e1.em();
9  B b1 = (A) d2;
```

(a) Explain if **Line 1** compiles.

(b) Explain if **Line 2** compiles.

(c) Explain if **Line 3** compiles. If yes, write down and explain how the output is printed. When tracing, consider only the earlier lines that compile.

(d) Explain if **Line 5** compiles. If yes, what are the static type and dynamic type of `d2` after `Line 5` is executed? When tracing, consider only the earlier lines that compile.

(e) Explain if **Line 6** compiles. If yes, write down and explain the output. When tracing, consider only the earlier lines that compile.

(f) Explain if **Line 7** compiles.

(g) Explain if **Line 8** compiles. If yes, write down and explain the output. If no, suggest a fix using type casting, then write down and explain how the output is printed. When tracing, consider only the earlier lines that compile.

(h) Explain why **Line 9** compiles.

But **Line 9** is problematic at runtime. Explain why and how we can extend the code to avoid it. When tracing, consider only the earlier lines that compile.

10. Consider the following classes, where we use `print` to abbreviate `System.out.println`:

```
interface I {
  void mi();
}
```

```
class A implements I {
  void mi() {
    println("A.mi"); }
}
```

```
class B implements I {
  void mi() {
    println("B.mi"); }
}
```

```
1  class Collector {
2    A[] as; int numberOfAs;
3    B[] bs; int numberOfBs;
4    Collector() {
5      as = new A[10]; bs = new B[10]; }
6    void addA(A a) {
7      as[numberOfAs] = a; numberOfAs++; }
8    void addB(B b) {
9      bs[numberOfBs] = b; numberOfBs++; }
10   void callAll() {
11     for(int i = 0; i < numberOfAs; i ++)
12     { as[i].mi(); }
13     for(int i = 0; i < numberOfBs; i ++)
14     { bs[i].mi(); }
15   }
16 }
```

```
1  class Tester {
2    static void main(String[] args) {
3      I i = new I();
4      B b = new B(); A a = new A();
5      Collector c = new Collector();
6      c.addB(b); c.addA(a);
7      c.callAll();
8    }
9  }
```

(a) Explain if the assignment `as[numberOfAs] = a` in **Line 7** of the above `Collector` class compiles.

(b) Explain if the method call `as[i].mi()` in **Line 12** of the above `Collector` class compiles.

(c) Explain if **Line 3** of the above `Tester` class compiles.

(d) Write and Explain the console output from **Line 7** of the above `Tester` class.

(e) The above `Collector` class does not make use of *polymorphism*, which results from the fact that classes `A` and `B` implement a common interface `I`. Rewrite the above `Collector` class, such that there is only one array attribute and one `add` method, and that the `callAll` method contains just a single loop.