Dec. 8

Exam Review Q&A

1. polymorphic array

2. polymorphic method param.

3. polymorphic return type

32 multiple choice (single choices)

2 * 32 = 64

4 written questions

36.

aliasing

type cast.

(C) obj.m()

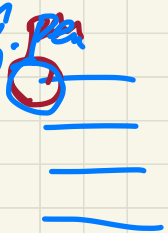
((C) obj).m()

implement a recursive method

↳ lecture examples

↳ Labs

↳ CodingBat



Tracing using a Stack

1. Caller/Callee
2. Recursive call Fib

```
class A extends G {
  A() { }
  void m() {
    print("A.m");
  }
}
```

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

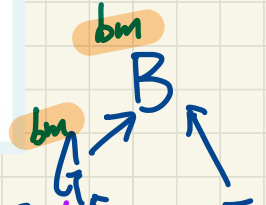
```
class C extends G {
  C() { }
  void bm(){
    print("C.bm");
  }
}
```

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

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

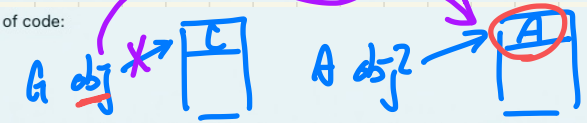
```
class F extends A {
  F() { }
  void bm(){
    print("F.bm");
  }
}
```

```
class G extends B {
  G() { }
  void bm(){
    print("G.bm");
  }
}
```



Consider the following lines of code:

```
G obj = new C();
A obj2 = new A();
obj = obj2;
```

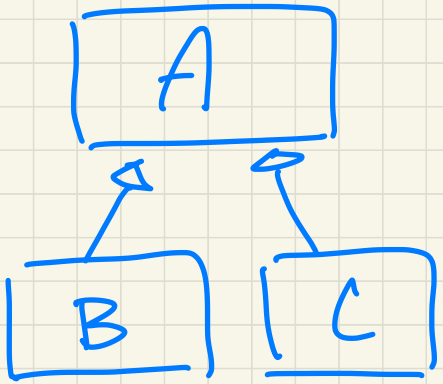


After executing the above lines of code, for each of the following method calls (involving type casts), choose the **best** description.

(F) obj.bm()	Choose...
((F) obj).bm()	Choose...
((A) obj).bm()	Choose...
((C) obj).bm()	Choose...
((F) obj).m()	Choose...
(A) obj.bm()	Choose...
((C) obj).m()	Choose...
(C) obj.bm()	Choose...
((A) obj).m()	Choose...

1. obj.bm() ✓ ST: G
2. cast RV to F
   ↳ Error : bm has RT void.

1. ((F) obj) compiles : downward
2. ((F) obj).bm()
   compilation : ✓
   runtime : CCE!




```
class Collection {
    A[] items;
    Collection() {...}
    void add(A item) {
        param.
    }
}
```

item = arg;

↳ ST of arg should be a descendant of the ST of item.

cast expr. has ST object.

① B obj = new B();
C.add(obj);

② C.add((Object)obj); X

```
class Application {
    Collection c = new Collection();
    c.add(arg);
}
```

argument

Assume a non-empty integer array **ns** of length 3 and an integer variable **i**.

Consider the following fragment of code:

```
if(0 > i || ns[i] % 2 == 1 || ns.length <= i) {
   System.out.println("Outcome 1");
}
else {
   System.out.println("Outcome 2");
}
```

When executing the above program, which of the following value or values of variable **i** will result in an **ArrayIndexOutOfBoundsException**?

- [ ] a. 4
- [ ] b. -2
- [ ] c. 1
- [ ] d. -1
- [ ] e. 3
- [ ] f. None of the listed answers is correct.
- [ ] g. 2
- [ ] h. 0

Consider the following classes:

```
public class A {
  private String att1;
  private C att2;
}
```

```
public class B {
  private String att3;
  private A[] att4;
}
```

```
public class C {
  private String att5;
  private B[] att6;
}
```

For each of the private attributes, we assume that the corresponding public accessor (with the appropriate return type) exists in the same class, e.g., accessor "public String getAtt1()" in class A, accessor "public A[] getAtt4()" in class B.

Say we are now in the context of some method in **class B**. For the following expressions (where i, j, and k are assumed to be integers), choose **all** that are **valid** (i.e., succeed to compile).

- [ ] a. att4[i].getAtt2().getAtt6()[j].att4[k].getAtt2().getAtt5()
- [ ] b. att4[i].getAtt2().att6[j].getAtt4()[k].getAtt1()
- [ ] c. att4[i].getAtt2().getAtt6()[j].getAtt4().getAtt1()
- [ ] d. att4[i].att2.getAtt6()[j].getAtt4()[k].getAtt1()
- [ ] e. att4[i].getAtt2().getAtt6().getAtt4()[j].getAtt1()
- [ ] f. att4[i].getAtt2().getAtt6()[j].getAtt4()[k].att1
- [ ] g. att4[i].getAtt2().getAtt6()[j].att4[k].getAtt1()
- [ ] h. att4[i].getAtt2().getAtt6()[j].getAtt4()[k].getAtt1()