1. For the following classes, A call to “new B()” prints BCAD (True/False)

```
class A{
      { System.out.print("A"); }
       static { System.out.print("B"); }
}
```

```
class B extends A {
       static { System.out.print("C"); }
       B(){ System.out.print("D"); }
}
```

2. In Java, string objects, including string literals, consume memories on the heap. (True/False)

3. `String a = "hi"; String b = "hi";` The statement: `System.out.print(a==b)` will print true; (True/False)

4. A method of a subclass can access the private members of its parent class. (True/False)

5. A subclass can change the modifier of an overridden method from “protected” to “public” because the latter is stronger in visibility (True/False)

6. For the following classes,

```
class X {
   static void a()
{ System.out.println("B");}
       X(){ a();}
}
```

```
class Y extends X {
   static void a()
{ System.out.println("A");}
}
```

A call to “new Y()” prints “B”. (True/False)

7. For the following classes,

```
class A{
       void a() { this.b();
System.out.print("A");}
                  void b()
{ System.out.print("B");}
       }
```

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

A call to “new B().a()” prints “CB”. (True/False)

8. A method foo calls another method bar that declares throwing exception. If foo doesn’t surround the call with try/catch blocks then foo also needs to declare throwing exception. (True/False)

9. The following program fails to compile. (True/False)

```
     public static void main(String s[]) {
                                    int x = 10;
              System.out.println(x instanceof Object);
}
```

10. For the following program, if call to test throws an IOException, it will print “Orange Fruit Meat” in order. (True/False)

```
try {
        test();
        print(“apple”);
        return;
} catch(IOException e1){
        System.out.print("Orange ");
        return;
} catch(Exception e2){
        System.out.print("Pineapple ");
        return;
} finally {
        System.out.print("Fruit ");
}
System.out.println("Meat");
```

11. When `Apple` is a subtype of `Fruit` and `Stack<T>` is a generic class, the statement `Stack<? super Apple> st = new Stack<Fruit>();` is correct. (True/False)

12. Is the following program correct? True/False

```
class M { }
public class Generics {
        public void doit(ArrayList<? extends M> list) {
                // omited
        }
  public void test() {
                ArrayList<M> list = new ArrayList<M>();
       doit(list);
        }

}
```

13. Is the right side the correct type-erasure of the left? (True/False)

`class Parent<T> {`
`  T compute(T t){return t;}`
`}`
`p = new Parent<String>();`
`String s = p.compute(“s”);`
`String s = p.compute(circle);`
	`class Parent{`
	`  Object compute(Object t){return t; }`
	`}`
	`p = new Parent();`
	`String s = (String) p.compute(“s”);`
	`String s = (String) p.compute(circle);`

14. The following program prints 2 (True/False)

```
class T {
   void m(String e) { //print 1}
}
class X<E> extends T {
  void m(E o) { //print 2 }
}
//main
X t = new X<String>();
```

```
t.m((Object)new String());
```

15. The following is a functional interface. (True/False)

```
interface T { int m(String s, int y); }
```

16. We learned the following functional interfaces in class:

A. Predicate<T>: *T -> Boolean,* **B.** Function<T, R>: *T -> R,* **C:** Consumer<T>: *T -> void,* **D:** Supplier<T>: *() -> T*

The usage of lambda expression : `factory(someclass::new)` requires the support of functional interface A. (True/False)

17. We learned the following functional interfaces in class:

A. Predicate<T>: *T -> Boolean,* **B.** Function<T, R>: *T -> R,* **C:** Consumer<T>: *T -> void,* **D:** Supplier<T>: *() -> T*

For the following function that accepts a lambda expression, the returning type should be C. (True/False)

```
RETURNINGTYPE compose (Predicate<X> p, Y y){
        return (X x)->{
                if(p.test(x)) return y;
                else return null;
        };
}
```

18. Deadlock will occur when a thread already hold a lock and another thread try to acquire the same one. (True/False)

19. Consider counting the number of "M" objects created, the following code has a race condition. (True/False)

```
class M {
        int count; {count++;}
}
public void test() {
        ExecutorService s = Executors.newCachedThreadPool();
        synchronized(s) {
                s.execute(M::new);
                s.execute(M::new);
        }
}
```

20. In the following program, the while loop in the first thread never terminates (exception code omitted). (True/False)

```
ExecutorService s =
Executors.newCachedThreadPool();
int fish = 0;
s.execute(()->{
   while(fish==0)
        synchronized(this) {
             wait();
             fish--;
        }
});
```

```
s.execute(()->{
        synchronized(this) {
           fish++;
           notify();
        }
});
```