Lambda

1. Fill in the blanks: The _____functional interface does not take any inputs, while _____ the functional interface does not return any data.
A. IntConsumer, LongSupplier
B. IntSupplier, Function
C. Supplier, DoubleConsumer
D. UnaryOperator, Consumer

2. Which functional interface takes a long value as an input argument and has an accept() method?
A. LongConsumer
B. LongFunction
C. LongPredicate
D. LongSupplier

3. What is the output of the following application?

```
package beach;
import java.util.function.*;
class Tourist {
public Tourist(double distance) {
this.distance = distance;
}
public double distance;
}
public class Lifeguard {
private void saveLife(Predicate<Tourist> canSave, Tourist tourist) {
System.out.print(canSave.test(tourist) ? "Saved" : "Too far"); // y1
}
public final static void main(String... sand) {
new Lifeguard().saveLife(s -> s.distance<4, new Tourist(2)); // y2
}
}
```

A. Saved
B. Too far
C. The code does not compile because of line y1.
D. The code does not compile because of line y2.

4. Which of the following statements about DoubleSupplier and Supplier<Double> is not true?
A. Both are functional interfaces.
B. Lambdas for both can return a double value.
C. Lambdas for both cannot return a null value.
D. One supports a generic type, the other does not.

5. Which functional interface, when filled into the blank, allows the class to compile?

```
package space;
import java.util.function.*;
public class Asteroid {
public void mine( _____lambda) {
// TODO: Apply functional interface
```

```
}
public static void main(String[] debris) {
new Asteroid().mine((s,p) -> s+p);
}
}
```
A. BiConsumer<Integer,Double>
B. BiFunction<Integer,Double,Double>
C. BiFunction<Integer,Integer,Double>
D. Function<Integer,Double>

6. Assuming the proper generic types are used, which lambda expression cannot be assigned to a ToDoubleBiFunction functional interface reference?
A. (Integer a, Double b) -> {int c; return b;}
B. (h,i) -> (long)h
C. (String u, Object v) -> u.length()+v.length()
D. (x,y) -> {int z=2; return y/z;}

7. Which of the following is not a functional interface in the java.util.function package?
A. BiPredicate
B. DoubleUnaryOperator
C. ObjectDoubleConsumer
D. ToLongFunction

8. What is the output of the following application?
```
package zoo;
import java.util.function.*;
public class TicketTaker {
private static int AT_CAPACITY = 100;
public int takeTicket(int currentCount, IntUnaryOperator<Integer> counter) {
return counter.applyAsInt(currentCount);
}
public static void main(String...theater) {
final TicketTaker bob = new TicketTaker();
final int oldCount = 50;
final int newCount = bob.takeTicket(oldCount,t -> {
if(t>AT_CAPACITY) {
throw new RuntimeException("Sorry, max has been reached");
}
return t+1;
});
System.out.print(newCount);
}
}
```
A. 51
B. The code does not compile because of lambda expression.
C. The code does not compile for a different reason.
D. The code compiles but prints an exception at runtime.

9. Which functional interface returns a primitive value?
A. BiPredicate
B. CharSupplier

C. LongFunction
D. UnaryOperator

10. Which functional interface, when entered into the blank below, allows the class to compile?

```
package groceries;
import java.util.*;
import java.util.function.*;
public class Market {
private static void checkPrices(List<Double> prices,_____scanner) {
prices.forEach(scanner);
}
public static void main(String[] right) {
List<Double> prices = Arrays.asList(1.2, 6.5, 3.0);
checkPrices(prices,
p -> {
String result = p<5 ? "Correct" : "Too high";
System.out.println(result);
});
}
}
```

A. Consumer
B. DoubleConsumer
C. Supplier<Double>
D. None of the above

11. Which of the following three functional interfaces is not equivalent to the other two?
A. BiFunction<Double,Double,Double>
B. BinaryOperator<Double>
C. DoubleFunction<Double>
D. None of the above. All three are equivalent.

12. Which lambda expression can be passed to the magic() method?

```
package show;
import java.util.function.*;
public class Magician {
public void magic(BinaryOperator<Long> lambda) {
lambda.apply(3L, 7L);
}
}
```

A. magic((a) -> a)
B. magic((b,w) -> (long)w.intValue())
C. magic((c,m) -> {long c=4; return c+m;})
D. magic((Integer d, Integer r) -> (Long)r+d)

13. What is the output of the following program?

```
package ai;
import java.util.function.*;
public class Android {
public void wakeUp(Supplier supplier) { // d1
supplier.get();
}
```

```
public static void main(String... electricSheep) {
Android data = new Android();
data.wakeUp(() -> System.out.print("Program started!")); // d2
}
}
```

A. Program started!
B. The code does not compile because of line d1 only.
C. The code does not compile because of line d2 only.
D. The code does not compile because of both lines d1 and d2.

14. Which statement about all UnaryOperator functional interfaces (generic and primitive) is correct?
A. The input type must be compatible with the return type.
B. Some of them take multiple arguments.
C. They each take a generic argument.
D. They each return a primitive value.

17. Fill in the blanks: In the Collection interface, the method removeIf() takes a _____ , while the method forEach() takes a _____ .
A. Function, Function
B. Predicate, Consumer
C. Predicate, Function
D. Predicate, UnaryOperator

18. What is the output of the following application?

```
package nesting;
import java.util.function.*;
public class Doll {
private int layer;
public Doll(int layer) {
super();
this.layer = layer;
}
public static void open(UnaryOperator<Doll> task, Doll doll) {
while((doll = task.accept(doll)) != null) {
System.out.print("X");
}
}
public static void main(String[] wood) {
open(s -> {
if(s.layer<=0) return null;
else return new Doll(s.layer—);
}, new Doll(5));
}
}
```

A. XXXXX
B. The code does not compile because of the lambda expression.
C. The code does not compile for a different reason.
D. The code compiles but produces an infinite loop at runtime.

19. Which functional interface has a get() method?
A. Consumer
B. Function
C. Supplier
D. UnaryOperator

21. Which statement about functional interfaces and lambda expressions is not true?
A. A lambda expression may be compatible with multiple functional interfaces.
B. A lambda expression must be assigned to a functional interface when it is declared.
C. A method can return a lambda expression in the form of a functional interface instance.
D. The compiler uses deferred execution to skip determining whether a lambda expression compiles or not.

22. Which expression is compatible with the IntSupplier functional interface?
A. () -> 1<10 ? "3" : 4
B. () -> {return 1/0;}
C. () -> return 4
D. System.out::print

23. What is the output of the following application?

```
package tps;
import java.util.*;
class Boss {
private String name;
public Boss(String name) {
this.name = name;
}
public String getName() {return name.toUpperCase();}
public String toString() {return getName();}
}
public class Initech {
public static void main(String[] reports) {
final List<Boss> bosses = new ArrayList(8);
bosses.add(new Boss("Jenny"));
bosses.add(new Boss("Ted"));
bosses.add(new Boss("Grace"));
bosses.removeIf(s -> s.equalsIgnoreCase("ted"));
System.out.print(bosses);
}
}
```

A. [JENNY, GRACE]
B. [tps.Boss@4218224c, tps.Boss@815f19a]
C. The code does not compile because of the lambda expression.
D. The code does not compile for a different reason.

24. Which of the following method references can be passed to a method that takes Consumer<Object> as an argument?
I. ArrayList::new
II. String::new
III. System.out::println
A. I only

B. I, II, and III
C. I and III
D. III only

25. Which of the following is a valid functional interface in the java.util.function package?
A. FloatPredicate
B. ToDoubleBiFunction
C. UnaryIntOperator
D. TriPredicate

26. Which functional interface, when filled into the blank, prevents the class from compiling?

```
package morning;
import java.util.function.*;
public class Sun {
public static void dawn(_____ sunrise) {}
public void main(String... rays) {
dawn(s -> s+1);
}
}
```

A. DoubleUnaryOperator
B. Function<String,String>
C. IntToLongFunction
D. UnaryOperator

27. Which functional interface does not have the correct number of generic arguments?
A. BiFunction<T,U,R>
B. DoubleFunction<T,R>
C. ToDoubleFunction<T>
D. ToIntBiFunction<T,U>

Answers

1. C. The Supplier functional interface does not take any inputs, while the Consumer functional interface does not return any data. This behavior extends to the primitive versions of the functional interfaces, making Option C the correct answer. Option A is incorrect because IntConsumer takes a value, while LongSupplier returns a value. Options B and D are incorrect because Function and UnaryOperator both take an input and produce a value.

2. A. The LongSupplier interface does not take any input, making Option D incorrect. It also uses the method name getAsLong(). The rest of the functional interfaces all take a long value but vary on the name of the abstract method they use. LongFunction contains apply() and LongPredicate contains test(), making Options B and C, respectively, incorrect. That leaves us with LongConsumer, which contains accept(), making Option A the correct answer.

3. A. The code compiles without issue, so Options C and D are incorrect. The value for distance is 2, which based on the lambda for the Predicate will result in a true expression, and Saved will be printed, making Option A correct.

4. C. Both are functional interfaces in the java.util.function package, making Option A true. The major difference between the two is that Supplier<Double> takes the generic type Double, while the other does not take any generic type and instead uses the primitive double. For this reason, Options B and D are true statements. For Supplier<Double> in Option B, remember that the returned double value can be implicitly cast to Double. Option C is the correct answer. Lambdas for Supplier<Double> can return a null value since Double is an object type, while lambdas for DoubleSupplier cannot; they can only return primitive double values.

5. B. The lambda (s,p) -> s+p takes two arguments and returns a value. For this reason, Option A is incorrect because BiConsumer does not return any values. Option D is also incorrect, since Function only takes one argument and returns a value. This leaves us with Options B and C, which both use BiFunction, which takes two generic arguments and returns a generic value. Option C is incorrect because the datatype of the unboxed sum s+q is int and int cannot be autoboxed or implicitly cast to Double. Option B is correct. The sum s+p is of type double, and double can be autoboxed to Double.

6. C. To begin with, ToDoubleBiFunction<T,U> takes two generic inputs and returns a double value. Option A is compatible because it takes an Integer and Double and returns a Double value that can be implicitly unboxed to double. Option B is compatible because long can be implicitly cast to double. While we don't know the data types for the input arguments, we know that some values, such as using Integer for both, will work. Option C cannot be assigned and is the correct answer because the variable v is of type Object and Object does not have a length() method. Finally, Option D is compatible. The variable y could be declared double in the generic argument to the functional interface, making y/z a double return value.

7. C. The BiPredicate interface takes two generic arguments and returns a boolean value. Next, DoubleUnaryOperator takes a double argument and returns a double value. Last, ToLongFunction takes a generic argument and returns a long value. That leaves Option C, which is the correct answer. While there is an ObjDoubleConsumer functional interface, which takes a generic argument and a double value and does not return any data, there is no such thing as ObjectDoubleConsumer. Remember that Object is abbreviated to Obj in all functional interfaces in java.util.function.

8. C. The code does not compile, so Options A and D are incorrect. The IntUnaryOperator functional interface is not generic, so the argument IntUnaryOperator<Integer> in the takeTicket() does not compile, making Option C the correct answer. The lambda expression compiles without issue, making Option B incorrect. If the generic argument <Integer> was dropped from the argument declaration, the class would compile without issue and output 51 at runtime, making Option A the correct answer.

9. A. Option A is the correct answer because BiPredicate takes two generic types and returns a primitive boolean value. Option B is incorrect, since CharSupplier does not exist in java.util.function. Option C is also incorrect, since LongFunction takes a primitive long value and returns a generic type. Remember, Java only includes primitive functional interfaces that operate on double, int, or long. Finally, Option D is incorrect. UnaryOperator takes a generic type and returns a generic value.

10. D. First off, the forEach() method requires a Consumer instance. Option C can be immediately discarded because Supplier<Double> does not inherit Consumer. For this same reason, Option B is also incorrect. DoubleConsumer does not inherit from Consumer. In this

manner, primitive functional interfaces cannot be used in the forEach() method. Option A seems correct, since forEach() does take a Consumer instance, but it is missing a generic argument. Without the generic argument, the lambda expression does not compile because the expression p<5 cannot be applied to an Object. The correct functional interface is Consumer<Double>, and since that is not available, Option D is the correct answer.

11. C. BiFunction<Double,Double,Double> and BinaryOperator<Double> both take two Double input arguments and return a Double value, making them equivalent to one another. On the other hand, DoubleFunction<Double> takes a single double value and returns a Double value. For this reason, it is different from the other two, making Option C correct and Option D incorrect.

12. B. BinaryOperator<Long> takes two Long arguments and returns a Long value. For this reason, Option A, which takes one argument, and Option D, which takes two Integer values that do not inherit from Long, are both incorrect. Option C is incorrect because the local variable c is re-declared inside the lambda expression, causing the expression to fail to compile. The correct answer is Option B because intValue() can be called on a Long object. The result can then be cast to long, which is autoboxed to Long.

13. C. The program does not compile, so Option A is incorrect. The Supplier functional interface normally takes a generic argument, although generic types are not strictly required since they are removed by the compiler. Therefore, line d1 compiles while triggering a compiler warning, and Options B and D are incorrect. On the other hand, line d2 does cause a compiler error, because the lambda expression does not return a value. Therefore, it is not compatible with Supplier, making Option C the correct answer.

14. A. The input type of a unary function must be compatible with the return type. By compatible, we mean identical or able to be implicitly cast. For this reason, Option A is the correct answer. Option B is incorrect since all of the UnaryOperator functional interfaces, generic or primitive, take exactly one value. Option C is incorrect because the primitive functional interfaces do not take a generic argument. Finally, Option D is incorrect. For example, the generic UnaryOperator<T> returns an Object that matches the generic type.

15. C. Remember that all Supplier interfaces take zero parameters. For this reason, the third value in the table is 0, making Options A and B incorrect. Next, DoubleConsumer and IntFunction each take one value, double and int, respectively. On the other hand, ObjDoubleConsumer takes two values, a generic value and a double, and returns void. For this reason, Option C is correct, and Option D is incorrect.

16. D. All Consumer functional interfaces have a void return type. For this reason, the first and last values in the table are both void, making Options A and B incorrect. IntFunction takes an int and returns a generic value, while LongSupplier does not take any values and returns a long value. For this reason, Option C is incorrect, and Option D is correct.

17. B. The removeIf() method requires a Predicate since it operates on a boolean result, making Option A incorrect. The forEach() method takes a Consumer and does not return any data, making Option B correct, and Options C and D incorrect.

18. C. The code does not compile, so Option A is incorrect. The lambda expression compiles without issue, making Option B incorrect. The task variable is of type UnaryOperator<Doll>, with

the abstract method apply(). There is no accept() method defined on that interface, therefore the code does not compile, and Option C is the correct answer. If the code was corrected to use the apply() method, the rest of it would compile without issue. At runtime, it would then produce an infinite loop. On each iteration of the loop, a new Doll instance would be created with 5, since the postdecrement (--) operator returns the original value of the variable, and that would make Option D the correct answer.

19. C. To begin with, Consumer uses accept(), making Option A incorrect. Next, Function and UnaryOperator use apply(), making Options B and D, respectively, incorrect. Finally, Supplier uses get(), making Option C the correct answer.

20. D. First off, Options A and B are incorrect because the second functions for both return a double or Double value, respectively. Neither of these values can be sent to a UnaryOperator<Integer> without an explicit cast. Next, Option C is incorrect. The first functional interface Function<Double,Integer> takes only one input, but the diagram shows two inputs for the first functional interface. That leaves us with Option D. The first functional interface BiFunction<Integer,Double,Integer> takes an int, which can be implicitly autoboxed to Integer, and a Double and returns an Integer. The next functional interface, BinaryOperator<Integer>, takes two Integer values and returns an Integer value. Finally, this Integer value can be implicitly unboxed and sent to IntUnaryOperator, returning an int. Since these behaviors match our diagram, Option D is the correct answer.

21. D. Options A, B, and C are true statements about functional interfaces. A lambda may be compatible with multiple functional interfaces, but it must be assigned to a functional interface when it is declared or passed as a method argument. Also, a method can be created with the return type that matches a functional interface, allowing a lambda expression to be returned. Option D is the correct answer. Deferred execution means the lambda expression is not evaluated until runtime, but it is compiled. Compiler errors in the lambda expression will prevent the code from compiling.

22. B. Option A is incorrect because the String "3" is not compatible with the return type int required for IntSupplier. Option B is the correct answer. Although this will result in a divide by zero issue at runtime, the lambda is valid and compatible with IntSupplier. Option C is incorrect because the lambda expression is invalid. The return statement is only allowed inside a set of brackets {}. Finally, Option D is incorrect. The method reference is used for Supplier, not Consumer, since it takes a value and does not return anything.

23. C. The lambda expression is invalid because the input argument is of type Boss, and Boss does not define an equalsIgnoreCase() method, making Option C the correct answer. If the lambda was corrected to use s.getName() instead of s, the code would compile and run without issue, printing [JENNY, GRACE] at runtime and making Option A the correct answer.

24. D. First of all, Consumer<Object> takes a single Object argument and does not return any data. The classes ArrayList and String do not contain constructors that take an Object, so neither of the first two statements are correct. The third statement does support an Object variable, since the System.out.println(Object) method exists. For these reasons, Option D is the correct answer.

25. B. The java.util.function package does not include any functional interfaces that operate on the primitive float, making Option A incorrect. Remember, Java only includes primitive functional

interfaces that operate on double, int, or long. Option B is correct because it is a valid functional interface. Option C is incorrect because there is no UnaryIntOperator functional interface. Note that there is one called IntUnaryOperator. Option D is incorrect. The java.util.function package does not include any tri- operators, although many are easy to write.

26. D. A lambda expression can match multiple functional interfaces. It matches DoubleUnaryOperator, which takes a double value and returns a double value. Note that the data type of s+1 is double because one of the operands, in this case s, is double. It also matches Function<String,String> since the (+) operator can be used for String concatenation. Finally, it matches IntToLongFunction since the int value s+1can be implicitly cast to long. On the other hand, the lambda expression is not compatible with UnaryOperator without a generic type. When UnaryOperator is used without a generic argument, the type is assumed to be Object. Since the (+) operator is not defined on Object, the code does not compile due to the lambda expression body, making Option D the correct answer. Note that if the lambda expression did not rely on the (+) operator, such as s -> s, then UnaryOperator would be allowed by the compiler, even without a generic type.

27. B. The BiFunction interface takes two different generic values and returns a generic value, taking a total of three generic arguments. Next, ToDoubleFunction takes exactly one generic value and returns a double value, requiring one generic argument. The ToIntBiFunction interface takes two generic values and returns an int value, for a total of two generic arguments. For these reasons, Options A, C, and D are incorrect. The correct answer is Option B. DoubleFunction takes a double value and returns a generic result, taking exactly one generic argument, not two.