総仕上げ問題（ダウンロード版）
問題

■試験番号：1Z0-809

■問題数：85問

■試験時間：150分

1. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。(1つ選択)

```
11. Stream.of("Blue", "Red", "Yellow", "Green")
12.     .mapToInt(color -> color.length()) // line n1
13.     .filter(len -> len > 3)
14.     .limit(3)
15.     .forEach(System.out::print);
```

A. 「BlueYellowGreen」が表示される
B. 「Red」が表示される
C. 「465」が表示される
D. 何も表示されない
E. // line n1の行でコンパイルエラーとなる

➡ A2

2. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。(1つ選択)

```
11. List<Integer> values = Arrays.asList(1, 2, 3);
12. values.stream()
13.     .map(n -> n + 1) // line n1
14.     .peek(System.out::print) // line n2
15.     .count(); // line n3
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. // line n3の行でコンパイルエラーとなる
D. 何も表示されない
E. 「234」が表示される
F. 「3」が表示される

➡ A2

3. クラス設計において抽象クラスの代わりにインタフェースを使用する理由として適切なものを選びなさい。（2つ選択）

A. 関連しないクラスが実装を提供することが予想される場合
B. 関連するクラス間で実装コードを共有したい場合
C. メンバーのアクセスレベルの規定にpublic以外の修飾子を使用したい場合
D. staticもしくはfinalではないフィールドを宣言したい場合
E. 多重継承のメリットを享受したい場合

➡ A2

4. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Runnable r1 = () -> System.out.println("Let's go"); // line n1
12. final CyclicBarrier barrier = new CyclicBarrier(3, r1); // line n2
13. Runnable r2 = () -> {
14.     System.out.println("Awaiting...");
15.     try {
16.         barrier.await();
17.     } catch (Exception e) { }
18. };
19.
20. new Thread(r2).start();
21. new Thread(r2).start();
22. new Thread(r2).start();
```

A. 「Let's go」が表示された後に「Awaiting...」が3回表示される
B. 「Awaiting...」が3回表示された後に「Let's go」が表示される
C. // line n1の行でコンパイルエラーとなる
D. // line n2の行でコンパイルエラーとなる

➡ A3

5. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. LocalDate ld = LocalDate.of(2016, 9, 15);
12. ld.withYear(2017);
13. ld.withDayOfMonth(25);
14. System.out.println(ld);
```

A. 「2016-09-15」が表示される
B. 「2017-09-25」が表示される
C. コンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A3

6. 以下に示すコードがある。コンパイルを成功させるための変更として正しいものを選びなさい。（2つ選択）

```
1.  public abstract class X {
2.
3.      X() { }
4.
5.      protected void doIt() {
6.          System.out.println("X");
7.      }
8.  }
```

```
1.  public class Y extends X {
2.
3.      private int a;
4.
5.      Y(int a) {
6.          // line n1
7.          this.a = a;
8.      }
9.
10.     public void doIt() {
11.         System.out.println("Y");
12.     }
13. }
```

※次ページに続く

```
 1. public class Z extends Y {
 2.
 3.     private int b;
 4.     private int c;
 5.
 6.     Z(int b, int c) {
 7.         // line n2
 8.         this.b = b;
 9.         this.c = c;
10.     }
11.
12.     void doIt() {
13.         System.out.println("Z");
14.     }
15. }
```

A. Xクラスの宣言からabstractを取り除く
B. Yクラスの// line n1の行を「super();」にする
C. YクラスのdoItメソッドのpublic修飾子を取り除く
D. Zクラスの// line n2の行を「super(b);」にする
E. ZクラスのdoItメソッドにpublic修飾子を付ける

➡ A4

7. **次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。(1つ選択)**

```
11. Stream<List<String>> s1 = Stream.of(
12.     Arrays.asList("1", "Eric"),
13.     Arrays.asList("2", null)
14. );
15. Stream<String> s2 = s1.flatMapToInt(x -> x.stream());
16. s2.forEach(System.out::print);
```

A. 「1Eric2null」が表示される
B. 「12」が表示される
C. 「Ericnull」が表示される
D. コンパイルエラーとなる
E. 実行時にNullPointerExceptionがスローされる

➡ A4

8. 2016年9月10日から2020年9月10日までの年数「4」を表示するために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
11. LocalDate ld1 = LocalDate.of(2016, Month.SEPTEMBER, 10);
12. LocalDate ld2 = LocalDate.of(2020, Month.OCTOBER, 10);
13.
14. // line n1
15. System.out.println(years);
```

A. `int years = Period.between(ld1, ld2).getYears();`
B. `Long years = Duration.between(ld1, ld2).ofYears();`
C. `Period years = Period.between(ld1, ld2).getYears();`
D. `Duration years = Duration.between(ld1, ld2).ofYears();`
E. `Instant years = new Instant().between(ld1, ld2).getYears();`

➡ A5

9. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. BiConsumer<Integer, Double, Integer> c = (i, d) -> i * d; // line n1
12. int result = c.apply(2, 1.2); // line n2
13. System.out.println(result);
```

A. 「2」が表示される
B. 「2.4」が表示される
C. // line n1の行でコンパイルエラーとなる
D. // line n2の行で実行時に例外がスローされる

➡ A5

10. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
 1. public class Outer {
 2.
 3.     private int value = 5;
 4.
 5.     class Inner {
 6.         private int value = new Outer().value + 5; // line n1
 7.     }
 8.
 9.     public static void main(String[] args) {
10.         System.out.println(new Outer().new Inner().value); // line n2
11.     }
12. }
```

A. 「5」が表示される
B. 「10」が表示される
C. // line n1の行でコンパイルエラーとなる
D. // line n2の行でコンパイルエラーとなる

➡ A6

11. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
1. public interface X { }
```

```
1. public enum E implements X { // line n1
2.     A, B, C;
3. }
```

```
11. System.out.print(E.A instanceof E);
12. System.out.print(E.B instanceof X);
13. System.out.print(E.C instanceof Enum);
```

A. 「falsefalsefalse」が表示される
B. 「truetruetrue」が表示される
C. 「truetruefalse」が表示される
D. 列挙型はインタフェースをimplementsできないため、// line n1の行でコンパイルエラーとなる

➡ A6

12. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Map<String, Integer> itemMap = new HashMap<>();
12. itemMap.put("A", 200);
13. itemMap.put("B", 150);
14. itemMap.put("C", 300);
15.
16. String result = itemMap.stream()
17.     .filter(m -> m.getValue() >= 200)
18.     .map(m -> m.getKey())
19.     .collect(Collectors.joining());
20. System.out.println(result);
```

A. 「AC」が表示される
B. 「200300」が表示される
C. コンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A6

13. java.util.functionパッケージの関数型インタフェースに関する説明として正しいものを選びなさい。（2つ選択）

A. Consumerインタフェースの抽象メソッドはacceptメソッドである
B. IntSupplierインタフェースの抽象メソッドはgetメソッドである
C. BinaryOperatorインタフェースの抽象メソッドはapplyメソッドである
D. UnaryOperatorインタフェースは、BinaryOperatorインタフェースのスーパーインタフェースである
E. BiFunctionインタフェースは、Functionインタフェースのサブインタフェースである
F. Predicateインタフェースは、型パラメータにおいて2つの型変数を宣言している

➡ A6

14. 以下のコードに関する説明として正しいものを選びなさい。（1つ選択）

```
 1. public class Example {
 2.
 3.     interface X { }
 4.     interface Y extends X { }
 5.     interface Z extends Y { }
 6.
 7.     static class A<T> {    }
 8.     static void doIt(A<? super Y> arg) {}
 9.
10.     public static void main(String[] args) {
11.         doIt(new A<X>()); // line n1
12.         doIt(new A<Y>()); // line n2
13.         doIt(new A<Z>()); // line n3
14.         doIt(new A()); // line n4
15.     }
16. }
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. // line n3の行でコンパイルエラーとなる
D. // line n4の行でコンパイルエラーとなる
E. いずれの行もコンパイルエラーにはならない

➡ A7

15. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
 1. public class ResourceA implements AutoCloseable {
 2.
 3.     public void open() {
 4.         System.out.print("Ao");
 5.     }
 6.
 7.     public void close() {
 8.         System.out.print("Ac");
 9.     }
10. }
```

```
 1. public class ResourceB implements AutoCloseable {
 2.
 3.     public void open() {
 4.         System.out.print("Bo");
 5.     }
 6.
 7.     public void close() {
 8.         System.out.print("Bc");
 9.     }
10. }
```

```
 1. public class Main {
 2.
 3.     public static void main(String[] args) {
 4.
 5.         try (ResourceA a = new ResourceA();
 6.              ResourceB b = new ResourceB()) {
 7.             System.out.print("M");
 8.         }
 9.     }
10. }
```

A. 「MAcBc」が表示される
B. 「MBcAc」が表示される
C. 「AoBoMAcBc」が表示される
D. 「AoBoMBcAc」が表示される

➡ A7

☐16. 以下のようなインタフェースがある。「#3$」と表示するためのコードとして正しいものを選びなさい。（1つ選択）

```
11. @FunctionalInterface
12. public interface Foo {
13.     public Function<String, String> doIt(Function<Integer, String> func);
14. }
```

A.
```
Foo foo = (f, x) -> "#" + f.apply(x.length());
System.out.println(foo.doIt(x -> x + "$").apply("ABC"));
```

B.
```
Foo foo = f -> x -> "#" + f.apply(x.length());
System.out.println(foo.doIt(x -> x + "$").apply("ABC"));
```

C.
```
Foo foo = (f, x) -> "#" + f.accept(x.length());
System.out.println(foo.doIt(x -> x + "$").accept("ABC"));
```

D.
```
Foo foo = f -> x -> "#" + f.accept(x.length());
System.out.println(foo.doIt(x -> x + "$").accept("ABC"));
```

➡ A7

☐17. 現在の協定世界時を表すjava.util.Dateオブジェクトを生成するコードとして正しいものを選びなさい。（1つ選択）

A.
```
OffsetDateTime utc = OffsetDateTime.now(ZoneId.UTC);
Date date = Date.of(utc);
```

B.
```
ZonedDateTime utc = ZonedDateTime.of(ZoneId.getUTC());
Date date = Date.of(utc.toEpoch());
```

C.
```
ZonedDateTime utc = ZonedDateTime.now(ZoneOffset.UTC);
Date date = Date.from(utc.toInstant());
```

D.
```
OffsetDateTime utc = OffsetDateTime.of(ZoneOffset.getUTC());
Date date = Date.from(utc);
```

➡ A9

18. 以下に示すMyListResourceBundleクラスで実装すべきメソッドとして正しいものを選びなさい。（1つ選択）

```
11. public class MyListResourceBundle extends ListResourceBundle {
12.
13. }
```

A.
```
public String[] getContents() {
    return new String[] {
        "Uno", "Dos", "Tres"
    };
}
```

B.
```
public List<String> getContents() {
    return Arrays.asList("Uno", "Dos", "Tres");
}
```

C.
```
public Object[][] getContents() {
    return new Object[][] {
        {"1", "Uno"}, {"2", "Dos"}, {"3", "Tres"}
    };
}
```

D.
```
public Map<String, String> getContents() {
    return new HashMap<String, String>() {
        {put("1", "Uno");}
        {put("2", "Dos");}
        {put("3", "Tres");}
    };
}
```

➡ A9

☐ 19. 以下と等価なコードを選びなさい。（1つ選択）

```
11. int x = IntStream.of(1, 2, 3).map(i -> 1).sum();
```

A. `int x = IntStream.of(1, 2, 3).total();`
B. `int x = IntStream.of(1, 2, 3).distinct();`
C. `long x = IntStream.of(1, 2, 3).count();`
D. `long x = IntStream.of(1, 2, 3).limit();`

➡ A10

☐ 20. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Map<String, Integer> map = new HashMap<>();
12. map.put("A", 3);
13. map.put("B", 1);
14. map.put("C", 2);
15.
16. Set<Map.Entry<String, Integer>> entrySet = map.entrySet();
17. List<Map.Entry<String, Integer>> entryList = new ArrayList<>(entrySet);
18. // line n1
19. Collections.sort(entryList, (e1, e2) -> e2.getValue() - e1.getValue());
20. // line n2
21.
22. System.out.println(entryList);
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. 「[A=3, C=2, B=1]」が表示される
D. 「[C=2, B=1, A=3]」が表示される

➡ A10

21. 以下の中から、「1.5」を表示するコードとして正しいものを選びなさい。（1つ選択）

A.
```
Function<Double> func = x -> x * 0.5;
System.out.println(func.apply(3));
```
B.
```
DoubleFunction<Integer> func = x -> x * 0.5;
System.out.println(func.apply(3));
```
C.
```
DoubleFunction<Double> func = x -> x * 0.5;
System.out.println(func.applyAsDouble(3));
```
D.
```
IntToDoubleFunction func = x -> x * 0.5;
System.out.println(func.applyAsDouble(3));
```

➡ A10

22. LocalDateTimeオブジェクトから値を取得するためのメソッドとして正しいものを選びなさい。（3つ選択）

A. getDate
B. getYear
C. getCalendar
D. getMonth
E. getDayOfMonth
F. getWeek

➡ A12

23. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。(1つ選択)

```
1. public interface Bar {
2.     Foo doIt(int i);
3. }
```

```
1. public class Foo {
2.
3.     Foo() {
4.         System.out.println("Foo");
5.     }
6.
7.     Foo(int i) {
8.         System.out.println("Foo(" + i + ")");
9.     }
10. }
```

```
11. Bar bar = Foo::new; // line n1
12. bar.doIt(10); // line n2
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. 「Foo(10)」が表示される
D. 「Foo()」と「Foo(10)」が表示される

➡ A12

24. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。(1つ選択)

```
11. String str = "Java8";
12. ToIntFunction<String> func = str::indexOf; // line n1
13. int x = func.applyAsInt("8"); //line n2
14. System.out.println(x);
```

A. 「4」が表示される
B. // line n1の行でコンパイルエラーとなる
C. // line n2の行でコンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A12

25. JDBCを使用したデータベース接続URLに必須の情報として正しいものを選びなさい。（2つ選択）

A. ポート番号
B. パスワード
C. ホスト名
D. ユーザー名
E. データベース名

➡ A13

26. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Predicate<Integer> p1 = i -> i % 2 == 0;
12. Predicate<Integer> p2 = i -> i > 3;
13. Arrays.asList(1, 2, 3, 4, 5).stream()
14.     .filter(p1.and(p2).negate()) // line n1
15.     .forEach(System.out::print);
```

A. 「4」が表示される
B. 「1235」が表示される
C. // line n1の行でコンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A13

27. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. List<Map<List<Integer>, List<String>>> list = new ArrayList<>();
12. Map<Integer, String> map = new HashMap<>();
13. list.add(null); // line n1
14. list.add(map); // line n2
15. list.forEach(System.out::print);
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. 「null{}{}」が表示される
D. 実行時に例外がスローされる

➡ A13

28. JDBC APIに関する説明として正しいものを選びなさい。(3つ選択)

A. データベースにおけるトランザクションの分離レベルは、DriverManagerクラスを使用して設定する
B. 1つのStatementオブジェクトから利用できるResultSetオブジェクトは常に1つだけである
C. データベースのメタデータを取得するには、Connectionインタフェースのget MetaDataメソッドを使用する
D. JDBC 4.0に準拠したドライバは、JARファイル内の「META-INF/services」ディレクトリ内にjava.sql.Driverファイルが必要である
E. クエリの結果セットのレコードに対するカーソルの操作は、ResultSetMetaDataインタフェースで宣言されているメソッドを使用する

➡ A14

29. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。(1つ選択)

```
11. Map<String, Integer> map = new HashMap<>();
12. map.put("Allan", 20);
13. map.put("Bob", 30);
14. map.put("Chris", 40);
15. map.putIfAbsent("Allan", 50);
16. map.remove("Bob", 30); // line n1
17. map.replaceAll((k, v) -> k.length() + v); // line n2
18. System.out.println(map);
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. 「{Chris=45, Allan=25}」が表示される
D. 「{Chris=45, Allan=55}」が表示される
E. 「{Bob=33, Chris=45, Allan=25}」が表示される
F. 実行時に例外がスローされる

➡ A14

30. 以下のコードを実行した場合に、常に「OK」を表示した後に「Main」を表示するために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
1. public class MyAction extends RecursiveAction {
2.
3.     @Override
4.     protected void compute() {
5.         System.out.println("OK");
6.     }
7. }
```

```
1. public class MyRunnable implements Runnable {
2.
3.     public void run() {
4.         System.out.println("OK");
5.     }
6. }
```

```
11. ForkJoinPool pool = new ForkJoinPool();
12. // line n1
13. System.out.println("Main");
```

A. `pool.invoke(new MyAction());`
B. `pool.submit(new MyAction());`
C. `pool.execute(new MyAction());`
D. `pool.compute(new MyAction());`

➡ A15

31. java.util.stream.Streamインタフェースで宣言されている、ストリーム内の最大要素を返すメソッドとして正しいものを選びなさい。（1つ選択）

A. `T max(Comparator<? extends T> c)`
B. `T max(Comparable<? super T> c)`
C. `Optional<T> max(Comparator<? super T> c)`
D. `Optional<T> max(Comparable<? extends T> c)`

➡ A15

32. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Map<String, String> map = new HashMap<>();
12. map.put("Y", "A");
13. map.put("X", "a");
14. map.entrySet().stream()
15.   .sorted(Map.Entry.comparingByValue(String.CASE_INSENSITIVE_ORDER))
16.   .forEach(System.out::println);
```

A. X=a
   Y=A
B. Y=A
   X=a
C. コンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A15

33. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Stream<String> words = Stream.of("one", "two", "three");
12. int len = words.map(String::length) // line n1
13.     .reduce(0, (x, y) -> x + y); // line n2
14. System.out.println(len);
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. 「11」が表示される
D. 実行時に例外がスローされる

➡ A16

34. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
1.  public class MyCloseable implements Closeable {
2.      public void close() throws IOException {
3.          System.out.println("MyCloseable");
4.      }
5.  }
```

```
1.  public class MyAutoCloseable implements AutoCloseable {
2.      public void close() throws Exception {
3.          System.out.println("MyAutoCloseable");
4.      }
5.  }
```

```
11. try (Closeable closeableImpl = new MyCloseable();
12.       AutoCloseable autoCloseableImpl = new MyAutoCloseable()) {
13. } catch (Exception e) {
14.     System.out.println("exception");
15. } finally {
16.     System.out.println("finally");
17. }
```

A. finally
B. MyCloseable
   MyAutoCloseable
   finally
C. MyAutoCloseable
   MyCloseable
   finally
D. exception

➡ A16

35. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. IntUnaryOperator f1 = i -> i * 2;
12. IntUnaryOperator f2 = i -> i + 2;
13. IntStream.of(0, 1, 2)
14.      .map(f1.compose(f2)) // line n1
15.      .forEach(System.out::print);
```

A. 「246」が表示される
B. 「468」が表示される
C. // line n1の行でコンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A16

36. 以下のコードの「// line n1」に置き換え可能なコードとして正しいものを選びなさい。（1つ選択）

```
11. long count = Stream.of("C", "A", "B", "A", "B", "C", "A")
12.      .filter(s -> s.equals("B")) // line n1
13.      .count();
```

A. .filter(Predicate.isEqual("B"))
B. .filter(Predicate::isEqual("B"))
C. .filter(Predicate.equals("B"))
D. .filter(Predicate::equals("B"))

➡ A16

37. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Duration d1 = ChronoUnit.YEARS.getDuration().multipliedBy(10); // line n1
12. Duration d2 = ChronoUnit.DECADES.getDuration(); // line n2
13. System.out.println(d1.equals(d2));
```

A. 「true」が表示される
B. 「false」が表示される
C. // line n1の行でコンパイルエラーとなる
D. // line n2の行でコンパイルエラーとなる

➡ A17

38. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。ただし、現在の日付は「2016年10月1日」とする。（1つ選択）

```
11. LocalDate ld = LocalDate.of(2016, Month.OCTOBER, 1);
12. MonthDay md = MonthDay.of(ld.getMonth(), ld.getDayOfMonth());
13. System.out.println(md.equals(MonthDay.from(LocalDate.now())));
```

A. 「true」が表示される
B. 「false」が表示される
C. コンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A17

39. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Stream<String> stream = Stream.of("A", "B", "C");
12. String str = stream.parallel().reduce((a, b) -> a + "-" + b); // line n1
13. System.out.println(str);  // line n2
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. 常に「A-B-C」が表示される
D. 実行するたびに表示結果は異なる

➡ A17

40. 以下のようなinput.txtファイルがある。

```
ABC
XYZ
```

次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. try (BufferedReader reader = new BufferedReader(new FileReader("input.txt"));
12.      BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"))) {
13.
14.     String str = null;
15.     while ((str = reader.read()) != null) {
16.         writer.write(str);
17.     }
18. } catch (Exception e) {
19.     e.printStackTrace();
20. }
```

A. output.txtファイルが作成され、以下の内容が書き込まれる
   ABC
   XYZ
B. output.txtファイルが作成され、以下の内容が書き込まれる
   ABCXYZ
C. コンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A18

41. java.timeパッケージで提供されている列挙型として正しいものを選びなさい。（2つ選択）

A. DayOfWeek
B. MonthOfYear
C. Month
D. Week

➡ A18

42. 以下と等価なコードを選びなさい。(1つ選択)

```
11. int total = IntStream.of(1, 2, 3).sum();
```

A. `int total = IntStream.of(1, 2, 3).reduce(x -> x)`
B. `int total = IntStream.of(1, 2, 3).reduce((x, y) -> x + y)`
C. `int total = IntStream.of(1, 2, 3).reduce(0, (x, y) -> x + y)`
D. `int total = IntStream.of(1, 2, 3).reduce(1, x -> x)`

➡ A18

43. Streamインタフェースのcollectメソッドの引数に指定可能なCollectorsクラスのメソッドとして正しいものを選びなさい。(2つ選択)

A. orderBy
B. groupingBy
C. combining
D. joining

➡ A19

44. ForkJoinPoolクラスに関する説明として正しいものを選びなさい。(2つ選択)

A. 他のExecutorServiceとは異なり、Work-stealingアルゴリズムを使用する
B. デフォルトの並列性レベルは使用可能なプロセッサの数と同じである
C. タスクを実行するためのメソッドはinvokeメソッドのみが提供されている
D. タスク間の協調や調整が必要な処理に適している

➡ A19

45. 「2016-09-01」と表示するために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
11. LocalDate ld1 = LocalDate.of(2016, Month.APRIL, 1);
12. // line n1
13. System.out.println(ld2);
```

A. `LocalDate ld2 = ld1.at(Month.SEPTEMBER);`
B. `LocalDate ld2 = ld1.of(ChronoUnit.MONTH, 9);`
C. `LocalDate ld2 = ld1.with(ChronoField.MONTH_OF_YEAR, 9);`
D. `LocalDate ld2 = ld1.ofMonth(9);`

➡ A19

46. 選択肢に記述されたリソース・バンドル用のファイルがすべて存在してアクセスが可能な場合に、以下のコードによって使用されるファイルとして正しいものを選びなさい。（1つ選択）

```
11. Locale.setDefault(new Locale("en", "US"));
12. ResourceBundle rb = ResourceBundle.getBundle("MyResources");
13. String key = rb.getString("key");
```

A. MyResources.properties
B. MyResources_en.propertieCs
C. MyResources_en_US.properties
D. 上記のいずれのファイルも使用されない

➡ A20

47. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Set<String> set = new HashSet<>();
12. set.add("A");
13. set.add("B");
14. Iterator<String> it = set.iterator();
15. set.add("C");
16. while (it.hasNext()) {
17.     System.out.print(it.next());
18. }
19. set.forEach(System.out::print);
```

A. 「ABABC」が表示される
B. 「ABCABC」が表示される
C. コンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A20

48. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Stream.of(1, 2, 3)
12.     .peek(System.out::print)
13.     .map(i -> i * i)
14.     .peek(System.out::print) // line n1
15.     .count(); // line n2
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. 何も表示されない
D. 「3」が表示される
E. 「112439」が表示される

➡ A20

49. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. public interface Greeting {
12.     public void sayHello(String name);
13. }
```

```
11. String name = "John";
12. Greeting g = n -> System.out.println("Hello, " + name);
13. name = "Eric";
14. g.sayHello(name);
```

A. 「Hello, John」が表示される
B. 「Hello, Eric」が表示される
C. コンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A20

50. 入力された引数をそのまま返すFunctionインタフェースのメソッドとして正しいものを選びなさい。（1つ選択）

A. toEqual
B. identity
C. pass
D. goThrough

➡ A21

51. 以下に示すようなBookクラスがある。priceの合計値「64.95」を表示するために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
 1. public class Book {
 2.
 3.     private String title;
 4.     private Double price;
 5.
 6.     public Book(String title, Double price) {
 7.         this.title = title;
 8.         this.price = price;
 9.     }
10.
11.     public String getTitle() { return title; }
12.     public void setTitle(String title) { this.title = title; }
13.     public Double getPrice() { return price; }
14.     public void setPrice(Double price) { this.price = price; }
15. }
```

```
11. Book[] books = {
12.     new Book("Java Cookbook", 25.20),
13.     new Book("Beginning Java 8", 22.98),
14.     new Book("Java Pocket Guide", 16.77)
15. };
16.
17. // line n1
18.
19. System.out.println(total);
```

A.
```
Double total = books.stream()
    .collect(Collectors.sum(Book::getPrice));
```

B.
```
Double total = books.stream()
    .reduce(Collectors.sumToDouble(book -> book.getPrice()));
```

C.
```
Double total = Arrays.stream(books)
    .collect(Collectors.summingDouble(Book::getPrice));
```

D.
```
Double total = Arrays.stream(books)
    .reduce(Collectors.summing(book -> book.getPrice()));
```

➡ A21

52. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. BinaryOperator<Integer> op = (i1, i2) -> i1 * i2; // line n1
12. BiFunction<Integer, Integer, Integer> func = op; // line n2
13. System.out.println(func.apply(2, 3));
```

A. 「6」が表示される
B. // line n1の行でコンパイルエラーとなる
C. // line n2の行でコンパイルエラーとなる
D. 実行時に例外ClassCastExceptionがスローされる

➡ A21

53. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Stream<String> stream = Stream.of("Charles", "Jackie", "Eduard");
12. System.out.println(
13.     stream.collect(Collectors.partitioningBy(s -> s.contains("ar")))
14. );
```

A. 「[Charles, Eduard]」が表示される
B. 「[[Jackie], [Charles, Eduard]]」が表示される
C. 「{Charles=true, Jackie=false, Eduard=true}」が表示される
D. 「{false=[Jackie], true=[Charles, Eduard]}」が表示される
E. コンパイルエラーとなる
F. 実行時に例外がスローされる

➡ A21

54. 「024」と表示するために、「/* insert code here */」に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
11. AtomicInteger ai = new AtomicInteger(0);
12. for (int i = 0; i < 3; i++) {
13.     System.out.print(/* insert code here */);
14. }
```

A. ai.getAndIncrement(2)
B. ai.getAndIncrement(x -> x + 2)
C. ai.getAndAccumulate(x -> x + 2)
D. ai.getAndAccumulate(2, (x, y) -> x + y)

➡ A22

55. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Predicate<Integer> p = x -> x > 5;
12. IntStream s = IntStream.of(3, 7, 1, 9, 5);
13. System.out.println(s.anyMatch(p)); // line n1
```

A. // line n1の行でコンパイルエラーとなる
B. 「true」が表示される
C. 「false」が表示される
D. 「7」と「9」が表示される
E. 実行時に例外がスローされる

➡ A22

☐ 56. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
 1. public class Example {
 2.
 3.     private static AtomicInteger counter = new AtomicInteger(0);
 4.
 5.     static class Inc extends Thread {
 6.         public void run() {
 7.             System.out.print(counter.incrementAndGet());
 8.         }
 9.     }
10.
11.     static class Dec extends Thread {
12.         public void run() {
13.             System.out.print(counter.decrementAndGet());
14.         }
15.     }
16.
17.     public static void main(String[] args) {
18.
19.         for (int i = 0; i < 3; i++) {
20.             new Inc().start();
21.             new Dec().start();
22.         }
23.     }
24. }
```

A. 常に「000000」が表示される
B. 常に「101010」が表示される
C. 常に「123210」が表示される
D. 表示される値は実行ごとに異なる

➡ A23

57. 以下の中から、コンパイル・実行可能なコードを選びなさい。（1つ選択）

A. `Supplier<LocalDate> now = LocalDate::now();`
B. `Supplier<LocalDate> now = () -> LocalDate::now;`
C. `Supplier<LocalDate> now = LocalDate::new;`
D. `Supplier<LocalDate> now = LocalDate::now;`

➡ A23

58. 以下のような列挙型がある場合に、コンパイル・実行が可能なコードとして正しいものを選びなさい。（2つ選択）

```
11. public enum Directions {
12.     EAST, WEST, SOUTH, NORTH
13. }
```

A. `System.out.println(Directions.EAST);`
B. `System.out.println(Directions.WEST.ordinal());`
C. `System.out.println(Directions.indexAt("SOUTH"));`
D. `System.out.println(Directions.NORTH.value());`

➡ A23

59. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. List<Integer> list1 = new ArrayList<>(Arrays.asList()); // line n1
12. List<?> list2 = new ArrayList<>(list1); // line n2
13. Integer i = list2.get(0); // line n3
14. System.out.println(i);
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. // line n3の行でコンパイルエラーとなる
D. 「null」が表示される
E. 実行時に例外がスローされる

➡ A24

☐ 60. 以下のようなinput.txtファイルがある。

```
ABC
XYZ
```

次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. try (BufferedReader br = new BufferedReader(new FileReader("input.txt"))) {
12.     System.out.print(br.readLine());
13.     br.mark();
14.     System.out.print(br.readLine());
15.     br.reset();
16.     System.out.print(br.readLine());
17. } catch (IOException e) {
18.     e.printStackTrace();
19. }
```

A. 「ABCXYZXYZ」が表示される
B. 「ABCABCXYZ」が表示される
C. コンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A24

☐61. 以下に示すFooインタフェースがある。

```
1. public interface Foo<T> {
2.     public T doIt(int i);
3. }
```

以下のコードの「// line n1」の部分に挿入できるコードとして正しいものを選びなさい。（1つ選択）

```
11. // line n1
12. String[] array = foo.doIt(3);
```

A. `Foo<String>[] foo = String::new;`
B. `Foo<String>[] foo = String::new[];`
C. `Foo<String[]> foo = String[]::new;`
D. `Foo<String[]> foo = String::new[];`

➡ A25

□62. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。(1つ選択)

```
 1. public class Producer implements Runnable {
 2.
 3.     private int value = 1;
 4.     private final BlockingQueue<Integer> queue;
 5.
 6.     Producer(BlockingQueue<Integer> queue) {
 7.         this.queue = queue;
 8.     }
 9.
10.     public void run() {
11.         try {
12.             while (true) {
13.                 Thread.sleep((int)(Math.random() * 1000));
14.                 System.out.println("putting " + value + " to the queue.");
15.                 queue.put(value++); // line n1
16.                 System.out.println("¥t" + queue);
17.             }
18.         } catch (InterruptedException e) {
19.             e.printStackTrace();
20.         }
21.     }
22. }
```

※次ページに続く

```
1.  public class Consumer implements Runnable {
2.
3.      private final BlockingQueue<Integer> queue;
4.
5.      Consumer(BlockingQueue<Integer> queue) {
6.          this.queue = queue;
7.      }
8.
9.      public void run() {
10.         try {
11.             while (true) {
12.                 Thread.sleep((int)(Math.random() * 5000));
13.                 System.out.println("taking " + queue.take() + " from the queue."); // line n2
14.             }
15.         } catch (InterruptedException e) {
16.             e.printStackTrace();
17.         }
18.     }
19. }
```

```
11. BlockingQueue<Integer> queue = new ArrayBlockingQueue<>(10);
12. new Thread(new Producer(queue)).start();
13. new Thread(new Consumer(queue)).start();
```

A. 明示的に終了するまでプログラムは正常に実行し続ける
B. // line n1の行で、実行時に常に例外がスローされる
C. // line n2の行で、実行時に常に例外がスローされる
D. キューが満杯の場合に、// line n1の行で実行時に例外がスローされる可能性がある
E. キューが空の場合に、// line n2の行で実行時に例外がスローされる可能性がある

➡ A25

Q36

☐ 63. 以下のようなEmployeeテーブルがある。

ID	NAME
1	Bill
2	John
3	Eric

次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. try (Connection conn = DriverManager.getConnection(URL, USER, PASS);
12.     Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE);
13.     ResultSet rs = stmt.executeQuery("SELECT * FROM employee")) {
14.     rs.afterLast();
15.     rs.previous();
16.     System.out.println(rs.getString(2));
17. } catch (SQLException e) {
18.     e.printStackTrace();
19. }
```

A. 「Bill」が表示される
B. 「John」が表示される
C. 「Eric」が表示される
D. コンパイルエラーとなる
E. 実行時に例外がスローされる

➡ A25

64. 「Hello」を表示するために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
1. public class Outer {
2.     static class Inner {
3.         public final String message = "Hello";
4.     }
5. }
```

```
1. public class Main {
2.     public static void main(String[] args) {
3.         System.out.println(
4.             // line n1
5.         );
6.     }
7. }
```

A. `Outer.Inner.message`
B. `new Outer.Inner().message`
C. `Outer.new Inner().message`
D. `new Outer().Inner.message`

➡ A26

65. 以下のコードでコンパイルエラーとなる行を選びなさい。（1つ選択）

```
11. @FunctionalInterface
12. public interface Foo {
13.     abstract static void x(); // line n1
14.     default void y() {}; // line n2
15.     void z(); // line n3
16.     class Bar {} // line n4
17. }
```

A. // line n1
B. // line n2
C. // line n3
D. // line n4

➡ A26

66. 以下のコードを実行した場合に「Example [t=OK, u=OK]」と表示するために、Exampleクラスで実装するメソッドのコードとして正しいものを選びなさい。（1つ選択）

```
 1. public class Example<T, U> {
 2.
 3.     private T t;
 4.     private U u;
 5.
 6.     public Example(T t, U u) {
 7.         this.t = t;
 8.         this.u = u;
 9.     }
10.
11.     public T getT() {
12.         return t;
13.     }
14.
15.     public U getU() {
16.         return u;
17.     }
18. }
```

```
11. Example<String, String> e2 = Example.<String>doIt("OK");
12. System.out.println(e2);
```

A.
```
public static <T, T> Example<T> doIt(T value) {
    return new Example<T, T>(value, value);
}
```

B.
```
public static <T> Example<T> doIt(T value) {
    return new Example<T, T>(value, value);
}
```

C.
```
public static <T> Example<T, T> doIt(T value) {
    return new Example<T, T>(value, value);
}
```

D.
```
public static <T, T> Example<T, T> doIt(T value) {
    return new Example<T, T>(value, value);
}
```

➡ A26

67. 以下のコードをコンパイル、実行した場合に「表示結果」とするために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

【表示結果】

```
Steve Miller
Steve House
Jack Johnson
Doug Watkins
Doug Lee
```

```
1.  public class Employee {
2.
3.      private String firstName;
4.      private String lastName;
5.
6.      public Employee(String fName, String lName) {
7.          firstName = fName;
8.          lastName = lName;
9.      }
10.
11.     String getFirstName() {
12.         return firstName;
13.     }
14.
15.     String getLastName() {
16.         return lastName;
17.     }
18.
19.     @Override
20.     public String toString() {
21.         return firstName + " " + lastName;
22.     }
23. }
```

※次ページに続く

```
11. Employee[] employees = {
12.     new Employee("Doug", "Lee"),
13.     new Employee("Steve", "House"),
14.     new Employee("Jack", "Johnson"),
15.     new Employee("Doug", "Watkins"),
16.     new Employee("Steve", "Miller")
17. };
18.
19. Comparator<Employee> sortByFirstName =
20.           (e1, e2) -> e1.getFirstName().compareTo(e2.getFirstName());
21. Comparator<Employee> sortByLastName =
22.           (e1, e2) -> e1.getLastName().compareTo(e2.getLastName());
23.
24. // line n1
```

A.
```
Stream.of(employees)
    .sorted(sortByFirstName.thenComparing(sortByLastName))
    .forEach(System.out::println);
```

B.
```
Stream.of(employees)
    .sorted(sortByFirstName.reversed().thenComparing(sortByLastName))
    .forEach(System.out::println);
```

C.
```
Stream.of(employees)
    .sorted(sortByFirstName.thenComparing(sortByLastName).reversed())
    .forEach(System.out::println);
```

D.
```
Stream.of(employees)
    .sorted(sortByFirstName.reversed()
    .thenComparing(sortByLastName).reversed())
    .forEach(System.out::println);
```

➡ A26

68. 次のプログラムをコンパイル、実行したときの表示結果として正しいものを選びなさい。(1つ選択)

```
 1. public class X implements AutoCloseable {
 2.
 3.     public void doIt() throws Exception {
 4.         System.out.println("X.doIt");
 5.         throw new Exception();
 6.     }
 7.
 8.     public void close() throws Exception {
 9.         System.out.println("closing X");
10.     }
11. }
```

```
 1. public class Y implements AutoCloseable {
 2.
 3.     public void doIt() {
 4.         System.out.println("Y.doIt()");
 5.     }
 6.
 7.     public void close() throws Exception {
 8.         System.out.println("closing Y");
 9.     }
10. }
```

```
11. public static void main(String[] args) {
12.
13.     try (X x = new X(); Y y = new Y()) {
14.         x.doIt();
15.         y.doIt();
16.     } catch (Exception e) {
17.         System.out.print("exception");
18.     }
19. }
```

A. X.doIt
closing Y
closing X
exception

B. X.doIt
closing X
closing Y
exception

C. X.doIt
closing X
exception

D. X.doIt
closing Y
exception

➡ A27

☐ 69. java.util.functionパッケージの関数型インタフェースとその抽象メソッドの組み合わせとして、誤っているものを選びなさい。（1つ選択）

A. Supplier<T>インタフェースとT getメソッド
B. Predicate<T>インタフェースとtest(T t)メソッド
C. Consumer<T>インタフェースとvoid accept(T t)メソッド
D. Function<T>インタフェースとvoid apply(T t)メソッド

➡ A27

70. 以下のようなsample.txtファイルがある。

```
Red Blue Red Green
Green Red
```

次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. try (Stream<String> lines = Files.lines(Paths.get("sample.txt"))) {
12.     lines
13.         .flatMap(line -> Arrays.stream(line.split(" ")))
14.         .collect(Collectors.groupingBy(word -> word, Collectors.counting()))
15.         .entrySet().stream().sorted(Entry.comparingByValue())
16.         .forEach(e -> System.out.print(e.getKey()));
17. } catch (IOException e) {
18.     e.printStackTrace();
19. }
```

A. 「RedBlueGreen」が表示される
B. 「RedGreenBlue」が表示される
C. 「BlueGreenRed」が表示される
D. 「BlueRedGreen」が表示される
E. 「GreenBlueRed」が表示される
F. 「GreenRedBlue」が表示される

➡ A27

71. リスト内の最大値「7」を表示するために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
11. List<Integer> list = Arrays.asList(3, 6, 7, 2, 4);
12. // line n1
13. System.out.println(max);
```

A. `Integer max = list.stream().max();`
B. `Integer max = list.stream().max().get();`
C. `Integer max = list.stream().max((i, j) -> i - j).get();`
D. `Integer max = list.stream().mapToInt(i -> i).max();`

➡ A28

☐72. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
 1. public class NullableObject {
 2.
 3.     private Optional<String> str;
 4.
 5.     public NullableObject(Optional<String> str) {
 6.         this.str = str;
 7.     }
 8.
 9.     public Optional<String> getStr() {
10.         return str;
11.     }
12. }
```

```
11. NullableObject obj = new NullableObject(Optional.ofNullable(null)); // line n1
12. Optional<String> str = obj.getStr();
13. str.ifPresent(System.out::println).orElse("Empty"); // line n2
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. 「Empty」が表示される
D. 「void」が表示される
E. 「null」が表示される

➡ A28

☐73. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. Arrays.asList("a;b;c", "", "x;y").stream()
12.     .flatMap(str -> Arrays.stream(str.split(";"))) // line n1
13.     .forEach(str -> System.out.print(str + ":"));
```

A. 「a:b:c::x:y:」が表示される
B. 「a:b:cx:y:」が表示される
C. 「abc:xy:」が表示される
D. // line n1の行でコンパイルエラーとなる
E. 実行時に例外がスローされる

➡ A28

☐74. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
11. List<List<List<String>>> list = Arrays.asList(
12.     Arrays.asList(
13.         Arrays.asList("A", "B"),
14.         Arrays.asList("C", "D", "E"),
15.         Arrays.asList("F")),
16.     Arrays.asList(
17.         Arrays.asList("e", "a", "d"),
18.         Arrays.asList("c", "f"))
19. );
20.
21. list.stream()
22.     .flatMap(Collection::stream)
23.     .filter(l -> l.size() > 2)
24.     .map(String::toLowerCase) // line n1
25.     .distinct() // line n2
26.     .forEach(System.out::print);
```

A. // line n1の行でコンパイルエラーとなる
B. // line n2の行でコンパイルエラーとなる
C. 「cdae」が表示される
D. 「cdeead」が表示される

➡ A28

☐75. 以下のコードを実行した場合に「[1, 3, 5]」と表示するために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
11. ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));
12. // line n1
13. System.out.println(list);
```

A. `list.removeAll(i -> i % 2 == 0);`
B. `list.removeOf(i -> i % 2 == 0);`
C. `list.removeIf(i -> i % 2 == 0);`
D. `list.remove(i -> i % 2 == 0);`

➡ A29

☐76. Java SE 8で導入された新しいDate and Time APIに関する説明として正しいものを選びなさい。（2つ選択）

A. UNIXタイムをベースに設計されている
B. ISO 8601をベースに設計されている
C. 日時を表現するクラスと日時を操作するクラスが分離されている
D. 日時を表現するクラスはImmutableでスレッド・セーフである
E. タイムゾーンを扱うクラスは提供されていない

➡ A29

77. 以下のコードをコンパイル、実行したときに、「55」を表示するために行う修正として正しいものを選びなさい。（2つ選択）

```
 1. public class Sum implements Callable<Integer> { // line n1
 2.
 3.     private Integer val;
 4.
 5.     public Sum(Integer i) {
 6.         this.val = i;
 7.     }
 8.
 9.     public Integer call() throws Exception { // line n2
10.         int sum = 0;
11.         for (long i = 1; i <= val; i++) {
12.             sum += i;
13.         }
14.         return sum;
15.     }
16. }
```

```
11. Callable<Integer> task = new Sum(10);
12. Executor executor = Executors.newSingleThreadExecutor(); // line n3
13. Future<Integer> future = executor.call(task); // line n4
14. System.out.println(future.get());
15. executor.shutdown();
```

A. // line n1の行を以下のように修正する
```
public class Sum implements Runnable<Integer> {
```
B. // line n2の行を以下のように修正する
```
public Integer submit() throws Exception {
```
C. // line n3の行を以下のように修正する
```
ExecutorService executor = Executors.newSingleThreadExecutor();
```
D. // line n4の行を以下のように修正する
```
Future<Integer> future = executor.submit(task);
```

➡ A29

78. マルチスレッド・アプリケーションにおいて発生する可能性のある問題として正しいものを選びなさい。(3つ選択)

A. レース・コンディション
B. リクエスト・フォージェリ
C. デッドロック
D. スターベーション
E. インジェクション

➡ A30

79. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。(1つ選択)

```
11. Map<Integer, String> map = new HashMap<>();
12. map.put(1, "B");
13. map.put(2, "A");
14. map.entrySet().stream()
15.   .sorted(Collections.reverseOrder(Map.Entry.comparingByKey())) // line n1
16.   .forEach(System.out::println);
```

A. 以下のように表示される
```
1=B
2=A
```
B. 以下のように表示される
```
2=A
1=B
```
C. // line n1の行でコンパイルエラーとなる
D. 実行時に例外がスローされる

➡ A30

80. try-with-resources文で使用可能なクラスとして正しいものを選びなさい。（1つ選択）

A.
```
public class MyResource implements Closeable {
    protected void close() /* throws IOException */ {
        // ...
    }
}
```

B.
```
public class MyResource implements Closeable {
    public void autoClose() /* throws IOException */ {
        // ...
    }
}
```

C.
```
public class MyResource implements AutoCloseable {
    void close() /* throws IOException */ {
        // ...
    }
}
```

D.
```
public class MyResource implements AutoCloseable {
    public void close() throws IOException {
        // ...
    }
}
```

➡ A31

81. 関数型インタフェースとして正しいものを選びなさい。（2つ選択）

A.
```
@FunctionalInterface
public interface Foo {
    void doIt();
}
```

B.
```
@FunctionalInterface
public interface Foo {
    void doIt();
    boolean equals(Object obj);
}
```

C.
```
@FunctionalInterface
public interface Foo {
    boolean equals(Object obj);
}
```

D.
```
@FunctionalInterface
public interface Foo {
}
```

➡ A31

82. 以下のコードを実行した場合に、「[java, oracle, 1z0-809]」と表示するために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
11. String str = Stream.of("java", "oracle", "1z0-809")
12. // line n1
13. System.out.println(str);
```

A. `.collect(Collectors.adding("[", "]"));`
B. `.collect(Collectors.joining(", ", "[", "]"));`
C. `.collect(Collectors.adapting("[", "]"));`
D. `.collect(Collectors.mixing(", ", "[", "]"));`

➡ A31

☐ 83. 次のプログラムをコンパイル、実行したときの結果として正しいものを選びなさい。（1つ選択）

```
1.  public class StringConcatenator {
2.
3.      public static String result = "";
4.
5.      public static void concatenate(String str) {
6.          result += str + " ";
7.      }
8.  }
```

```
11. String[] words = "Write_Once_Run_Anywhere".split("_");
12. Arrays.stream(words)
13.     .parallel()
14.     .forEach(StringConcatenator::concatenate); // line n1
15. System.out.println(StringConcatenator.result);
```

A. // line n1の行でコンパイルエラーとなる
B. 「Write Once Run Anywhere」が表示される
C. 表示される内容は実行するたびに異なる
D. 実行時に例外がスローされる

➡ A32

84. データベースへ接続してConnectionオブジェクトを返すメソッドを完成するために、「// line n1」の部分に挿入するコードとして正しいものを選びなさい。（1つ選択）

```
11. public static Connection connectToDb() throws SQLException {
12.
13.     String url = "jdbc:mysql://localhost:3306/";
14.     String database = "test";
15.     String user = "root";
16.     String password = "mysql";
17.     // line n1
18. }
```

A. return DatabaseManager.getConnection(url, database, user, password);
B. return Connection.getConnection(url, database, user, password);
C. return DriverManager.getConnection(url + database, user, password);
D. return DatabaseDriver.getConnection(url + database, user, password);

➡ A32

☐ 85. 以下に示すディレクトリ構造において、次のプログラムの実行結果として正しいものを選びなさい。（1つ選択）

```
11. try (Stream<Path> stream = Files.list(Paths.get("dir"))) {
12.     System.out.println(
13.         stream.count()
14.     );
15. } catch (IOException e) {
16.     e.printStackTrace();
17. }
```

【ディレクトリ構造】

```
dir
├── a.txt
├── b.txt
├── sub
│     │
      ├── x.txt
      ├── y.txt
```

A. 「3」が表示される
B. 「4」が表示される
C. 「5」が表示される
D. 「6」が表示される
E. コンパイルエラーとなる

➡ A32