1 Exam Questions

1. What is printed by the following code snippet?

```
1 int birb = 1 + 2 * 5>=2 ? 4 : 2;
2 int mammals = 3 < 3 ? 1 : 5>=5 ? 9 : 7;
3 System.out.print(birb+mammals+"");
```

   A. `49`

   B. `13`

   C. `18`

   D. `99`

   E. It does not compile

Answer: B.
The code compiles without issue, so Option E is incorrect. For this problem, it helps to remember that + and * have a higher precedence than the ternary ? : operator. In the first expression, `1 + 2 * 5`is evaluated first, resulting in a reduction to `11>=2 ? 4 : 2`, and then `birb` being assigned a value of 4. In the second expression, the first ternary expression evaluates to false resulting in a reduction to the second right-hand expression `5>=5 ? 9 : 7`, which then assigns a value of 9 to `mammals`. In the `print()` statement, the first + operator is an addition operator, since the operands are numbers, resulting in the value of 4 + 9, 13. The second + operator is a concatenation since one of the two operands is a `String`. The result 13 is printed, making Option B the correct answer.

2. Which of the following statements about objects, reference types, and casting are correct?

   A. An object can be assigned to an inherited interface reference variable without an explicit cast.

   B. The compiler can prevent all explicit casts that lead to an exception at runtime.

   C. Casting an object to a reference variable does not modify the object in memory.

   D. An object can be assigned to a subclass reference variable without an explicit cast.

   E. An object can be assigned to a superclass reference variable without an explicit cast.

   F. An implicit cast of an object to one of its inherited types can sometimes lead to a `ClassCastException` at runtime.

Answer: A, C, E.
An object can be cast to a superclass or inherited interface type without an explicit cast. Furthermore, casting an object to a reference variable does not modify the object in any way; it just may change what methods and variables are immediately accessible. For these reasons, Options A, C, and E are correct. Option B is incorrect; since the compiler can try to block or warn about invalid casts, it cannot prevent them. For example, any object can be implicitly cast to `java.lang.Object`, then explicitly cast to any other object, leading to a `ClassCastException` at runtime. Option D is also incorrect because assigning an object to a subclass reference variable requires an explicit cast. Finally, Option F is incorrect. An object can always be cast to one of its inherited types, superclass or interface, without a `ClassCastException` being thrown.

3. What is the output of the following when run as `java WhatAClass seed flower plant`?

```
1 package unix;
2 import java.util.*;
3 public class WhatAClass {
4   public static void main(String[] args) {
5     int result = Arrays.binarySearch(args, args[0]);
6     System.out.println(result);
7   }
8 }
```

A. `0`

B. `1`

C. `2`

D. The code does not compile.

E. The code compiles but throws an exception at runtime.

F. The output is not guaranteed.

Answer: F.
The array is not sorted. It does not meet the pre-condition for a binary search. Therefore, the output is not guaranteed and the answer is Option F.

4. How many objects are eligible for garbage collection at the end of the `main()` method?

```
1  package store;
2  public class Primates {
3    static String primate1 = new String("oranghutan");
4    static String primate2 = new String("lemur");
5    public static void primateMethod() {
6      String primate3 = new String("gorilla");
7      primate2 = primate1;
8      primate1 = primate3;
9    }
10   public static void main(String... args) {
11     primateMethod();
12   }
13 }
```

A. `0`

B. `1`

C. `2`

D. `3`

E. The code does not compile

Answer: B.
While `primate3` goes out of scope after the `primateMethod()` method, the `gorilla` object is referenced by `primate1` and therefore cannot be garbage collected. Similarly, the `oranghutan` object is now referenced by `primate2`. No variables reference the `lemur` object, so it is eligible to be garbage collected, and Option B is correct.

5. Fill in the blanks: The _____ keyword is used in method declarations, the _____ keyword is used to guarantee a statement will execute even if an exception is thrown, and the _____ keyword is used to throw an exception to the surrounding process.

    A. `throw, finally, throws`

    B. `throws, catch, throw`

    C. `catch, finally, throw`

    D. `finally, catch, throw`

    E. `throws, finally, throw`

Answer: E.
The `throws` keyword is used in method declarations, while the `throw` keyword is used to throw an exception to the surrounding process, and the `finally` keyword is used to add a statement that is guaranteed to execute even if an exception is thrown. For these reasons, Option E is the correct answer.

6. Which statements best describe the result of this code?

```
1 package asean;
2 public class FlyingCar {
3   public static void main(String... args) {
4     String[] aseanTourLoops = new String[] { "Malaysia", "
    Indonesia", "Philippines" };
5     String[] times = new String[] { "Day", "Night" };
6     for (int i = 0, j = 0; i < aseanTourLoops.length; i++, j++)
7       System.out.println(aseanTourLoops[i] + " " + times[j]);
8   }
9 }
```

A. The `println` causes one line of output.

B. The `println` causes two lines of output.

C. The `println` causes three lines of output.

D. The code terminates successfully.

E. The code throws an exception at runtime.

Answer: B, E.
The first two iterations through the loop complete successfully, making Option B correct. However, the two arrays are not the same size and the for loop only checks the size of the first one. The third iteration throws an `ArrayIndexOutOfBoundsException`, making Option E correct.

7. Fill in the blanks: Because of _____, it is possible to _____ a method, which allows Java to support _____.

   A. abstract methods, override, inheritance

   B. concrete methods, overload, inheritance

   C. virtual methods, overload, interfaces

   D. inheritance, abstract, polymorphism

   E. virtual methods, override, polymorphism.

Answer: E.
For this question, it helps to try all answers out. Most of them do not make any sense. For example, overloading a method is not a facet of inheritance. Likewise, concrete and abstract methods can both be overridden, not just one. The only answer that is valid is Option E. Without virtual methods, overriding a method would not be possible, and Java would not truly support polymorphism.

8. What is the result of the following?

```
1  package calendar;
2  public class Seasons {
3    public static void seasons(String... names) {
4      int l = names[1].length(); // s1
5      System.out.println(names[l]); // s2
6    }
7    public static void main(String[] args) {
8      seasons("Summer", "Fall", "Winter", "Spring");
9    }
10 }
```

A. `Fall`

B. `Spring`

C. The code does not compile

D. The code throws an exception on line `s1`

E. The code throws an exception on line `s2`

Answer: E.
The code does compile. Line `s1` is a bit tricky because `length` is used for an array and `length()` is used for a `String`. Line `s1` stores the length of the `Fall` in a variable, which is `4`. Line s2 throws an `ArrayIndexOutOfBoundsException` because `4` is not a valid index for an array with four elements. Remember that indices start counting with zero. Therefore, Option E is correct.

9. How many lines of the following application contain compilation errors?

```
1  package percussion;
2
3  interface BadInterface {}
4  abstract class Abstract implements BadInterface {
5      public Abstract(int stones) {}
6      public void throwRock() {}
7  }
8  public class Concrete extends Abstract {
9      public void throwRock(int count) {}
10     public void fight() {
11         super.throwRock(5);
12     }
13     public static void main(String[] stones) {
14         BadInterface mn = new Concrete();
15         mn.fight();
16     }
17 }
```

A. None. The code compiles and runs without issue.

B. 1

C. 2

D. 3

E. 4

Answer: D.
The code definitely does not compile, so Option A is incorrect. The first problem with this code is that the `Concrete` class is missing a constructor causing the class declaration on line 8 to fail to compile. The default no-argument constructor cannot be inserted if the superclass, Abstract, does not define a no-argument constructor. The second problem with the code is that line 11 does not compile, since it calls `super.throwRock(5)`, but the version of `throwRock()` in the parent class does not take any arguments. Finally, line 15 does not compile. While `mn` may be a reference variable that points to a `Concrete()` object, the `fight()` method cannot be called unless it is explicitly cast back to a `Concrete` reference. For these three reasons, the code does not compile, and Option D is the correct answer.

10. What is the output of the following code?

```
1  package fly;
2  public class Penguin {
3    public int adjustFlippers(int length, String[] type) {
4      length++;
5      type[0] = "LONG";
6      return length;
7    }
8    public static void main(String[] climb) {
9      final Penguin p = new Penguin();
10     int length = 5;
11     String[] type = new String[1];
12     length = p.adjustFlippers(length, type);
13     System.out.print(length+","+type[0]);
14   }
15 }
```

A. `5,LONG`

B. `6,LONG`

C. `5,null`

D. `6,null`

E. The code does not compile

F. The code compiles but throws an exception at runtime.

Answer: B.
The application compiles and runs without issue, so Options E and F are incorrect. Java uses pass-by-value, so even though the change to length in the first line of the `adjustFlippers()` method does not change the value in the `main()` method, the value is later returned by the method and used to reassign the `length` value. The result is that `length` is assigned a value of `6`, due to it being returned by the method. For the second parameter, while the `String[]` reference cannot be modified to impact the reference in the calling method, the data in it can be. Therefore, the value of the first element is set to `LONG`, resulting in an output of `6,LONG`, making Option B the correct answer.

11. Examine the following code and select the correct statement (choose 1 option).

```
1 class StringBuilders {
2     public static void main(String... args) {
3         StringBuilder sb1 = new StringBuilder("eLion");
4         String jaud = null;
5         jaud = sb1.append("X").substring(sb1.indexOf("L"), sb1.
              indexOf("X"));
6         System.out.println(jaud);
7     }
8 }
```

A. The code will print `LionX`

B. The code will print `Lion`

C. The code will print `Lion` if line 5 is changed to the following:

```
jaud = sb1.append("X").substring(sb1.indexOf('L'), sb1.
   indexOf('X'));
```

D. The code will compile only when line 4 is changed to the following:

```
StringBuilder jaud = null;
```

Answer: B.
Option (a) is incorrect and option (b) is correct. The substring method doesn't include the character at the end position in the result that it returns. Hence, the code prints `Lion`. Option (c) is incorrect. If line 5 is changed as suggested in this option, the code won't compile. You can't pass a char to `StringBuilder`'s method `indexOf`; it accepts `String`. Option (d) is incorrect because there are no compilation issues with the code.

12. Given the following code,

```
interface Jumpable {
  int height = 1;
  default void worldRecord() {
    System.out.print(height);
  }
}
interface Moveable {
  int height = 2;
  static void worldRecord() {
    System.out.print(height);
  }
}

class Kangaroo implements Jumpable, Moveable {
  int height = 3;
  Kangaroo() {
    worldRecord();
  }
  public static void main(String args[]) {
    Jumpable j = new Kangaroo();
    Moveable m = new Kangaroo();
    Chair c = new Kangaroo();
  }
}
```

what is the output? Select 1 option.

A. `111`

B. `123`

C. `333`

D. `222`

E. Compilation error

F. Runtime exception

Answer: A.
The constructor of the class `Chair` invokes the default non-static method defined in the interface `Jumpable`. Moreover, if only the `static worldRecord()` method in the interface `Moveable` were defined, its invocation would have to be qualified (that is, `Moveable.worldRecord();`) for the class `Chair` to compile.

13. Given the following code, which option, if used to replace `/* INSERT CODE HERE */`, will enable the class `Jungle` to determine whether the reference variable `animal` refers to an object of the class `Penguin` and print `1`? (Select 1 option.)

```
class Animal{ float age; }
class Penguin extends Animal { int beak;}
class Jungle {
  public static void main(String args[]) {
    Animal animal = new Penguin();
    /* INSERT CODE HERE */
    System.out.println(1);
  }
}
```

A. `if (animal instanceof Penguin)`

B. `if (animal instanceOf Penguin)`

C. `if (animal == Penguin)`

D. `if (animal = Penguin)`

Answer: A.
Option (b) is incorrect because the correct operator name is `instanceof` and not `instanceOf` (note the capitalised O). Options (c) and (d) are incorrect. Neither of these lines of code will compile because they are trying to compare and assign a class name to a variable, which isn't allowed.

14. Given that the file `Test.java`, which defines the following code, fails to compile, select the reasons for the compilation failure (choose 2 options).

```
class Human {
  Human(String value) {}
}
class Michael extends Human {}
class Test {
  public static void main(String args[]) {
    Michael m = new Michael();
  }
}
```

A. The class `Human` fails to compile.

B. The class `Michael` fails to compile.

C. The default constructor can call only a no-argument constructor of a base class.

D. The code that creates the object of the class `Michael` in the class `Test` did not pass a `String` value to the constructor of the class `Michael`.

Answer: B, C.
The class `Michael` doesn't compile, so the class `Test` can't use a variable of type `Michael`, and it fails to compile. While trying to compile the class `Michael`, the Java compiler generates a default constructor for it, which looks like the following:

```
Michael() {
    super();
}
```

Note that a derived class constructor must always call a base class constructor. When Java generates the previous default constructor for the class `Michael`, it fails to compile because the base class doesn't have a no-argument constructor. The default constructor that's generated by Java can only define a call to a no-argument constructor in the base class. It can't call any other base class constructor.

15. Examine the following code and select the correct statements (choose 2 options).

```
class Bottle {
  void Bottle() {}
  void Bottle(MayonnaiseBottle w) {}
}
class MayonnaiseBottle extends Bottle {}
```

A. A base class can't pass reference variables of its defined class as method parameters in constructors.

B. The class compiles successfully—a base class can use reference variables of its derived class as method parameters.

C. The class `Bottle` defines two overloaded constructors.

D. The class `Bottle` can access only one constructor.

Answer: B, D.
A base class can use reference variables and objects of its derived classes. Note that the methods defined in the class `Bottle` aren't constructors but regular methods with the name `Bottle`. The return type of a constructor isn't void.

16. Given the following code, which option, if used to replace `/* INSERT CODE HERE */`, will cause the code print `110`? (Select 1 option.)

```
class Book {
  private int pages = 100;
}
class EBook extends Book {
  private int interviews = 2;
  private int totalPages() { /* INSERT CODE HERE */ }
  public static void main(String[] args) {
    System.out.println(new EBook().totalPages());
  }
}
```

A. `return super.pages + this.interviews*5;`

B. `return this.pages + this.interviews*5;`

C. `return super.pages + interviews*5;`

D. `return pages + this.interviews*5;`

E. None of the above

Answer: E.
The variable pages has `private` access in the class `Book`, and it can't be accessed from outside this class.

17. Given the following code,

```
class NoMoneyException extends Exception {}
class Pen{
  void write(String val) throws NoMoneyException {
    int c = (10 - 7)/ (8 - 2 - 6);
  }
  void article() {
    //INSERT CODE HERE
  }
}
```

which of the options, when inserted at `//INSERT CODE HERE`, will define a valid use of the method `write` in the method `article`? (Select 2 options.)

A.
```
try {
    new Pen().write("story");
  } catch (NoMoneyException e) {}
```

B.
```
try {
    new Pen().write("story");
  } finally {}
```

C.
```
try {
    write("story");
  } catch (Exception e) {}
```

D.
```
try {
    new Pen().write("story");
  } catch (RuntimeException e) {}
```

Answer: A, C.
On execution, the method write will always throw an `ArithmeticException` (a `RuntimeException`) due to division by 0. But this method declares to `throw` a `NoMoneyException`, which is a checked exception. Because `NoMoneyException` extends the class `Exception` and not `RuntimeException`, `NoMoneyException` is a checked exception. When you call a method that throws a checked exception, you can either handle it using a `try-catch` block or declare it to be `thrown` in your method signature. Option (a) is correct because a call to the method write is enclosed within a `try` block. The `try` block is followed by a `catch` block, which defines a handler for the exception `NoMoneyException`. Option (b) is incorrect. The call to the method write is enclosed within a `try` block, followed by a `finally` block. The `finally` block isn't used to handle an exception. Option (c) is correct. Because `NoMoneyException` is a subclass of `Exception`, an exception handler for the class `Exception` can handle the exception `NoMoneyException` as well. Option (d) is incorrect. This option defines an exception handler for the class `RuntimeException`. Because `NoMoneyException` is not a subclass of `RuntimeException`, this code won't handle `NoMoneyException`.

18. What is the output of the following code? (Select 1 option.)

```
class JammyJellyfish {
  static String name = "m1";
  void ubuntuName() {
    String name = "m2";
    if (8 > 2) {
      String name = "m3";
      System.out.println(name);
    }
  }
  public static void main(String[] args) {
    JammyJellyfish m1 = new JammyJellyfish();
    m1.ubuntuName();
  }
}
```

A. `m1`

B. `m2`

C. `m3`

D. The code fails to compile

Answer: D.
The class `JammyJellyfish` defines three variables with the name `name`: The code fails to compile due to the definition of two local variables with the same name (`name`) in the method `ubuntuName`. If this code were allowed to compile, the scope of these local variables would overlap—the variable name defined outside the if block would be accessible to the complete method `ubuntuName`. The scope of the local variable `name`, defined within the if block, would be limited to the if block. Within the if block, how do you think the code would differentiate between these local variables? Because there's no way to do so, the code fails to compile.

19. What is the output of the following code? (Select 1 option.)

```
class JaBowl {
  public static void main(String args[]) {
    String jasFood = "Corn";
    System.out.println(jasFood);
    mix(jasFood);
    System.out.println(jasFood);
  }
  static void mix(String foodIn) {
    foodIn.concat("A");
    foodIn.replace('C', 'B');
  }
}
```

A. `Corn`
   `BornA`

B. `Corn`
   `CornA`

C. `Corn`
   `Born`

D. `Corn`
   `Corn`

Answer: D.
`String` objects are immutable. This implies that using any method can't change the value of a `String` variable. In this case, the `String` object is passed to a method, which seems to, but doesn't, change the contents of `String`.

20. What is the output of the following code? (Select 1 option.)

```
class SwJava {
  public static void main(String args[]) {
    String[] shapes = {"Circle", "Square", "Triangle"};
    switch (shapes) {
      case "Square": System.out.println("Circle"); break;
      case "Triangle": System.out.println("Square"); break;
      case "Circle": System.out.println("Triangle"); break;
    }
  }
}
```

A. The code prints `Circle`

B. The code prints `Square`

C. The code prints `Triangle`

D. The code prints
```
Circle
Square
Triangle
```

E. The code prints
```
Triangle
Circle
Square
```

F. The code fails to compile

Answer: F.
The question tries to trick you; it passes a `String[]` value to a switch construct by passing it an array of `String` objects. The code fails to compile because an array isn't a valid argument to a switch construct. The code would have compiled if it passed an element from the array shapes (shapes[0], shapes[1], or shapes[2]).

21. Given the following definition of the classes `Human`, `Father`, and `Home`, which option, if used to replace `//INSERT CODE HERE`, will cause the code to compile successfully? (Select 3 options.)

```
class Human {}
class Father extends Human {
  public void dance() throws ClassCastException {}
}
class Home {
  public static void main(String args[]) {
    Human p = new Human();
    try {
      ((Father)p).dance();
    }
    //INSERT CODE HERE
  }
}
```

A.
```
catch (NullPointerException e) {}
catch (ClassCastException e) {}
catch (Exception e) {}
catch (Throwable t) {}
```

B.
```
catch (ClassCastException e) {}
catch (NullPointerException e) {}
catch (Exception e) {}
catch (Throwable t) {}
```

C.
```
catch (ClassCastException e) {}
catch (Exception e) {}
catch (NullPointerException e) {}
catch (Throwable t) {}
```

D.
```
catch (Throwable t) {}
catch (Exception e) {}
catch (ClassCastException e) {}
catch (NullPointerException e) {}
```

E. `finally`

Answer: A, B, E.
Because `NullPointerException` and `ClassCastException` don't share a base class–derived class relationship, these can be placed before or after each other. The class `Throwable` is the base class of `Exception`. Hence, the exception handler for the class `Throwable` can't be placed before the exception handler of the class `Exception`. Similarly, Exception is a base class for `NullPointerException` and `ClassCastException`. Hence, the exception handler for the class `Exception` can't be placed before the exception handlers of the class `ClassCastException` or `NullPointerException`. Option (e) is OK because no checked exceptions are defined to be thrown.

22. What is the output of the following code? (Select 1 option.)

```
import java.time.*;
class Camera {
  public static void main(String args[]) {
    int hours;
    LocalDateTime now = LocalDateTime.of(2020, 10, 01, 0 , 0);
    LocalDate before = now.toLocalDate().minusDays(1);
    LocalTime after = now.toLocalTime().plusHours(1);
    while (before.isBefore(after) && hours < 4) {
      ++hours;
    }
    System.out.println("Hours:" + hours);
  }
}
```

A. The code prints `Camera:null`.

B. The code prints `Camera:Adjust settings manually`

C. The code prints `Camera:`.

D. The code will fail to compile.

Answer: D.
Note the type of the variables now, before, and after—they aren't the same. The code fails compilation because the code `before.isBefore(after)` calls the `isBefore` method on an instance of `LocalDate`, passing it a `LocalTime` instance, which isn't allowed. The local variable hours isn't initialised prior to being referred to in the while condition (`hours < 4`), which is another reason why the class doesn't compile.

23. The output of the class `TestJaJavaCourse`, defined as follows, is `300`:

```
class Course {
  int enrollments;
}
class TestJaJavaCourse {
  public static void main(String args[]) {
    Course c1 = new Course();
    Course c2 = new Course();
    c1.enrollments = 100;
    c2.enrollments = 200;
    System.out.println(c1.enrollments + c2.enrollments);
  }
}
```

What will happen if the variable `enrollments` is defined as a `static` variable? (Select 1 option.)

A. No change in output. `TestJaJavaCourse` prints `300`.

B. Change in output. `TestJaJavaCourse` prints `200`.

C. Change in output. `TestJaJavaCourse` prints `400`.

D. The class `TestJaJavaCourse` fails to compile.

Answer: C.
The code doesn't fail compilation after the definition of the variable `enrollments` is changed to a `static` variable. A `static` variable can be accessed using a variable reference of the class in which it's defined. All the objects of a class share the same copy of the static variable. When the variable `enrollments` is modified using the reference variable `c2`, `c1.enrollments` is also equal to `200`. Hence, the code prints the result of 200 + 200, that is, `400`.

24. What is the output of the following code? (Select 1 option.)

```
String jaudStr[] = new String[][]{{null},new String[]{"a","b","c
   "},{new String()}}[0] ;
String jaudStr1[] = null;
String jaudStr2[] = {null};
System.out.println(jaudStr[0]);
System.out.println(jaudStr2[0]);
System.out.println(jaudStr1[0]);
```

A. `null`
   `NullPointerException`

B. `null`
   `null`
   `NullPointerException`

C. `NullPointerException`

D. `null`
   `null`
   `null`

Answer: B.
The trickiest assignment in this code is the assignment of the variable `jaudStr`. The following line of code may seem to (but doesn't) assign a two-dimensional `String` array to the variable `jaudStr`:

```
String jaudStr[] = new String[][]{{null},new String[]{"a","b
   ","c"},{new String()}}[0] ;
```

The preceding code assigns the first element of a two-dimensional `String` array to the variable `jaudStr`. The following indented code will make the previous statement easier to understand:

```
String jaudStr[] = new String[][]{
     {null}, // First element of two-dimensional array-an
        array of length 1
     new String[]{"a","b","c"}, // Second element of two-
       dimensional array - an array of length 3
    {new String()} // // Third element of twodimensional array
        - an array of length 1
        } // End of definition of 2d array
    [0] ; // 1st element of this array {null}
```

So let's look at the simplified assignment:

```
String jaudStr[] = {null};
String jaudStr1[] = null;
String jaudStr2[] = {null};
```

Revisit the code that prints the array elements:

```
System.out.println(jaudStr[0]); // prints null
System.out.println(jaudStr2[0]); // prints null
System.out.println(jaudStr1[0]); // throws
   NullPointerException
```

Because `jaudStr` refers to an array of length 1 (`{null}`), `jaudStr[0]` prints `null`. `jaudStr2` also refers to an array of length 1 (`{null}`), so `jaudStr2[0]` also prints `null`. `jaudStr1` refers to `null`, not to an array. An attempt to access the first element of `jaudStr1` throws a `NullPointerException`.

25. Examine the following code and select the correct statement (choose 1 option).

```
1  import java.util.*;
2  class Human {}
3  class Emp extends Human {}
4  class TestArrayList {
5    public static void main(String[] args) {
6      ArrayList<Object> list = new ArrayList<>();
7      list.add(new String("1234")); //LINE1
8      list.add(new Human()); //LINE2
9      list.add(new Emp()); //LINE3
10     list.add(new String[]{"abcd", "xyz"}); //LINE4
11     list.add(LocalDate.now().plus(1)); //LINE5
12   }
13 }
```

A. The code on line 1 won't compile.

B. The code on line 2 won't compile.

C. The code on line 3 won't compile.

D. The code on line 4 won't compile.

E. The code on line 5 won't compile.

F. None of the above.

G. All the options from (A) through (E).

Answer: E.
The type of an `ArrayList` determines the type of the objects that can be added to it. An `ArrayList` can add to it all the objects of its derived class. Options (a) to (d) will compile because the class `Object` is the superclass of all Java classes; the `ArrayList` list as defined in this question will accept all types of objects, including arrays, because they are also objects. Although a `LocalDate` instance can be added to an `ArrayList`, the code in option (e) won't compile. `LocalDate.now()` returns a `LocalDate` instance. But the class `LocalDate` doesn't define a `plus()` method, which accepts an integer value to be added to it—there's actually one plus method that accepts a `TemporalAmount` instance. You can use any of the following methods to add days, months, weeks, or years to LocalDate, passing it long values:

```
plusDays(long daysToAdd)
plusMonths(long monthsToAdd)
plusWeeks(long weeksToAdd)
plusYears(long yearsToAdd)
```

You can also use the following method to add a duration of days to `LocalDate`, passing it an instance of `Period (Period implements TemporalAmount)`:

```
plus(TemporalAmount amountToAdd)
```

26. Examine the following code and select the correct statement (choose 1 option).

```
public class If2 {
  public static void main(String args[]) {
    int a = 10; int b = 20; boolean c = false;
    if (b > a) if (++a == 10) if (c!=true) System.out.println(1);
    else System.out.println(2); else System.out.println(3);
  }
}
```

A. 1

B. 2

C. 3

D. No output

Answer: C.
The key to answering questions about unindented code is to indent it. Here's how:

```
if (b > a)
    if (++a == 10)
        if (c!=true)
            System.out.println(1);
        else
            System.out.println(2);
    else System.out.println(3);
```

Now the code becomes much simpler to look at and understand. Remember that the last `else` statement belongs to the inner `if (++a == 10)`. As you can see, `if (++a == 10)` evaluates to `false`, the code will print `3`.

2 Programming Exercises

1. Create a Java program that takes a user-inputted string and performs the following tasks:
    1. Reverse the string: Reverse the order of characters in the string.
    2. Check for Palindrome: Determine if the string is a palindrome (reads the same backward as forward).
    3. Count Vowels: Count the number of vowels (a, e, i, o, u) in the string.
    4. Extract Email Addresses: Extract all email addresses from the string using regular expressions.

Answer:

```
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class StringManipulation {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter a string: ");
    String inputString = scanner.nextLine();

    // Reverse the string
    String reversedString = new StringBuilder(inputString).
   reverse().toString();
    System.out.println("Reversed string: " + reversedString);

    // Check for palindrome
    if (inputString.equalsIgnoreCase(reversedString)) {
      System.out.println("The string is a palindrome.");
    } else {
      System.out.println("The string is not a palindrome.");
    }

    // Count vowels
    int vowelCount = 0;
    for (char c : inputString.toCharArray()) {
      if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c ==
    'u' ||
          c == 'A' || c == 'E' || c == 'I' || c == 'O' || c ==
    'U') {
        vowelCount++;
      }
    }
    System.out.println("Number of vowels: " + vowelCount);
```

```
    // Extract email addresses
    Pattern pattern = Pattern.compile("\\b[\\w.-]+@[\\w
   .-]+\\.\\w{2,4}\\b");
    Matcher matcher = pattern.matcher(inputString);
    while (matcher.find()) {
      System.out.println("Email address found: " + matcher.
   group());
    }
  }
}
```

2. Create a Java program to simulate a restaurant management system. The system should have the following classes:

    1. `MenuItem`: This class should represent a menu item with properties like `name`, `price`, and `category` (e.g., appetizer, main course, dessert).
    2. `Order`: This class should represent a customer's order, including a list of menu items and the total cost.
    3. `Waiter`: This class should represent a waiter who can take orders, process payments, and deliver food.
    4. `Kitchen`: This class should simulate the kitchen, receiving orders from waiters, preparing food, and notifying waiters when orders are ready.

    Tasks

    1. Create Menu: Implement the MenuItem class and create a menu with various items, categorized as appetizers, main courses, and desserts.
    2. Take Orders: The `Waiter` class should allow customers to order items from the menu.
    3. Process Orders: The `Waiter` should send orders to the `Kitchen` class.
    4. Prepare Food: The `Kitchen` class should simulate food preparation time and notify the `Waiter` when an order is ready.
    5. Deliver Orders: The `Waiter` should deliver the prepared food to the customer.
    6. Calculate Bill: The `Waiter` should calculate the total bill for the customer's order, including taxes and any discounts.

    Optional:

    1. Error Handling: Implement error handling for situations like out-of-stock items or invalid orders.
    2. User Interface: Create a simple text-based user interface to interact with the system.
    3. Data Structures: Use appropriate data structures like `ArrayLists` or `HashMaps` to store menu items, orders, and customer information.
    4. Concurrency: Consider using threads to simulate concurrent order processing and food preparation.

Answer:
MenuItem.java

```
public class MenuItem {
  private String name;
  private double price;
  private String category;
  // Constructor and getters/setters
```

```
  public MenuItem(String name, double price, String category)
   {
    this.name = name;
    this.price = price;
    this.category = category;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public double getPrice() {
    return price;
  }

  public void setPrice(double price) {
    this.price = price;
  }

  public String getCategory() {
    return category;
  }

  public void setCategory(String category) {
    this.category = category;
  }


}
```

Order.java

```
import java.util.ArrayList;
import java.util.List;

public class Order {
  private List<MenuItem> items;
  private double totalCost;

  public Order() {
    items = new ArrayList<>();
    totalCost = 0;
  }
```

```
  public void addItem(MenuItem item) {
    items.add(item);
    totalCost += item.getPrice();
  }

  public List<MenuItem> getItems() {
    return items;
  }

  public double getTotalCost() {
    return totalCost;
  }
}
```

Waiter.java

```
public class Waiter {
  private Kitchen kitchen;

  public Waiter(Kitchen kitchen) {
    this.kitchen = kitchen;
  }

  public void takeOrder(Customer customer) {
    // ... (take order details, create Order object)
    kitchen.receiveOrder(order);
  }

  public void deliverOrder(Order order) {
    // ... (deliver food to customer)
  }
}
```

Kitchen.java

```
public class Kitchen {
  public void receiveOrder(Order order) {
    // Simulate cooking time
    try {
      Thread.sleep(5000); // 5 seconds
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    // Notify the waiter that the order is ready
    waiter.deliverOrder(order);
  }
```

```
}
```

Main Class

```
public class Restaurant {
  public static void main(String[] args) {
    // Create menu items
    MenuItem pizza = new MenuItem("Pizza", 10.99, "Main Course
   ");
    // ... (other menu items)

    // Create a waiter and kitchen
    Waiter waiter = new Waiter();
    Kitchen kitchen = new Kitchen();
    waiter.setKitchen(kitchen);

    // Simulate a customer order
    Customer customer = new Customer();
    waiter.takeOrder(customer);
  }
}
```

3. Create a Java program that reads a text file containing a list of words, one word per line. The program should then:

    1. Count word frequencies: Count the frequency of each word in the file.
    2. Sort words by frequency: Sort the words by their frequency, descending order.
    3. Write the sorted word frequencies to a new file.

Answer:

```
import java.io.*;
import java.util.*;

public class WordFrequencyCounter {
  public static void main(String[] args) {
    String inputFileName = "input.txt";
    String outputFileName = "output.txt";

    try (BufferedReader reader = new BufferedReader(new
   FileReader(inputFileName))) {
      Map<String, Integer> wordCounts = new HashMap<>();
      String line;

      while ((line = reader.readLine()) != null) {
        wordCounts.put(line, wordCounts.getOrDefault(line, 0)
   + 1);
      }

      // Sort words by frequency, descending order
      List<Map.Entry<String, Integer>> sortedWords = new
   ArrayList<>(wordCounts.entrySet());
      sortedWords.sort(Collections.reverseOrder(Map.Entry.
   comparingByValue()));

      // Write the sorted word frequencies to a new file
      try (BufferedWriter writer = new BufferedWriter(new
   FileWriter(outputFileName))) {
        for (Map.Entry<String, Integer> entry : sortedWords) {
          writer.write(entry.getKey() + ": " + entry.getValue
   () + "\n");
        }
      }
    } catch (IOException e) {
      System.err.println("Error reading or writing file: " + e
   .getMessage());
    }
  }
```

```
}
```

4. Create a Java program that simulates a library catalog. The program should store a list of books with their titles and ISBN numbers. The user should be able to:

    1. Add new books to the catalog
    2. Search for a book by title or ISBN using binary search
    3. Display a list of all books in the catalog

Answer:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

class Book implements Comparable<Book> {
  private String title;
  private String ISBN;

  public Book(String title, String ISBN) {
    this.title = title;
    this.ISBN = ISBN;
  }

  public String getTitle() {
    return title;
  }

  public String getISBN() {
    return ISBN;
  }

  // Compares two books based on their titles for sorting
  @Override
  public int compareTo(Book other) {
    return this.title.compareTo(other.title);
  }
}

public class LibraryCatalog {
  private List<Book> books;

  public LibraryCatalog() {
    books = new ArrayList<>();
  }

  public void addBook(Book book) {
```

```
    books.add(book);
    Collections.sort(books); // Sort the books after adding a
 new one
  }

  public Book searchByTitle(String title) {
    for (Book book : books) {
      if (book.getTitle().equalsIgnoreCase(title)) {
        return book;
      }
    }
    return null;
  }

  public Book searchByISBN(String ISBN) {
    // Use binary search for efficient searching
    int index = Collections.binarySearch(books, new Book(ISBN,
    ""), (b1, b2) -> b1.getISBN().compareTo(b2.getISBN()));
    if (index >= 0) {
      return books.get(index);
    }
    return null;
  }

  public void displayBooks() {
    for (Book book : books) {
      System.out.println("Title: " + book.getTitle() + ", ISBN
 : " + book.getISBN());
    }
  }

  public static void main(String[] args) {
    LibraryCatalog catalog = new LibraryCatalog();

    Scanner scanner = new Scanner(System.in);
    while (true) {
      System.out.println("1. Add Book");
      System.out.println("2. Search by Title");
      System.out.println("3. Search by ISBN");
      System.out.println("4. Display All Books");
      System.out.println("5. Exit");
      System.out.print("Enter your choice: ");

      int choice = scanner.nextInt();
      scanner.nextLine(); // Consume newline character
```

```
      switch (choice) {
        case 1:
          System.out.print("Enter book title: ");
          String title = scanner.nextLine();
          System.out.print("Enter ISBN: ");
          String ISBN = scanner.nextLine();
          catalog.addBook(new Book(title, ISBN));
          break;
        case 2:
          System.out.print("Enter book title to search: ");
          title = scanner.nextLine();
          Book bookByTitle = catalog.searchByTitle(title);
          if (bookByTitle != null) {
            System.out.println("Book found: " + bookByTitle.
  getTitle() + ", ISBN: " + bookByTitle.getISBN());
          } else {
            System.out.println("Book not found.");
          }
          break;
        case 3:
          System.out.print("Enter ISBN to search: ");
          ISBN = scanner.nextLine();
          Book bookByISBN = catalog.searchByISBN(ISBN);
          if (bookByISBN != null) {
            System.out.println("Book found: " + bookByISBN.
  getTitle() + ", ISBN: " + bookByISBN.getISBN());
          } else {
            System.out.println("Book not found.");
          }
          break;
        case 4:
          catalog.displayBooks();
          break;
        case 5:
          System.exit(0);
        default:
          System.out.println("Invalid choice.");
      }
    }
  }
}
```