Java

Q1. Given the string "strawberries" saved in a variable called fruit, what would `fruit.substring(2, 5)` return?

- ☐ rawb
- ☒ raw
- ☐ awb
- ☐ traw

Reasoning: The substring method accepts two arguments.

- The first argument is the index to start(includes that char at 2)
- and the second the index of the string to end the substring(excludes the char at 5).
- Strings in Java are like arrays of chars.
- Therefore, the method will return "raw" as those are the chars in indexes 2,3 and 4.
- You can also take the ending index and subtract the beginning index from it, to determine how many chars will be included in the substring (5-2=3).

Q2. How can you achieve runtime polymorphism in Java?

- ☐ method overloading
- ☐ method overrunning
- ☒ method overriding
- ☐ method calling

Q3. Given the following definitions, which of these expressions will NOT evaluate to true? `boolean b1 = true, b2 = false; int i1 = 1, i2 = 2;`

- ☐ `(i1 | i2) == 3`
- ☒ `i2 && b1`
- ☐ `b1 || !b2`
- ☐ `(i1 ^ i2) < 4`

Reasoning: i2 && b1 are not allowed between int and boolean.

Q4. What is the output of this code?

```
class Main {
  public static void main (String[] args) {
    int array[] = {1, 2, 3, 4};
    for (int i = 0; i < array.size(); i++) {
       System.out.print(array[i]);
    }
  }
}
```

☒ It will not compile because of line 4.
☐ It will not compile because of line 3.
☐ 123
☐ 1234

Reasoning: array.size() is invalid, to get the size or length of the array array.length can be used.

Q5. Which of the following can replace the CODE SNIPPET to make the code below print "Hello World"?

```
interface Interface1 {
    static void print() {
        System.out.print("Hello");
    }
}

interface Interface2 {
    static void print() {
        System.out.print("World!");
    }
}
```

☐ `super1.print(); super2.print();`
☐ `this.print();`
☐ `super.print();`
☒ `Interface1.print(); Interface2.print();`

Reference

Q6. What does the following code print?

```
String str = "abcde";
str.trim();
str.toUpperCase();
str.substring(3, 4);
System.out.println(str);
```

☐ CD
☐ CDE
☐ D
☒ "abcde"

Reasoning: You should assign the result of trim back to the String variable. Otherwise, it is not going to work, because strings in Java are immutable.

Q7. What is the result of this code?

```java
class Main {
    public static void main (String[] args){
        System.out.println(print(1));
    }
    static Exception print(int i){
        if (i>0) {
            return new Exception();
        } else {
            throw new RuntimeException();
        }
    }
}
```

- ☐ It will show a stack trace with a runtime exception.
- ☒ "java.lang.Exception"
- ☐ It will run and throw an exception.
- ☐ It will not compile.

Q8. Which class can compile given these declarations?

```java
interface One {
    default void method() {
        System.out.println("One");
    }
}

interface Two {
    default void method () {
        System.out.println("One");
    }
}
```

- ☐ A

```java
class Three implements One, Two {
    public void method() {
        super.One.method();
    }
}
```

- ☐ B

```java
class Three implements One, Two {
    public void method() {
        One.method();
    }
}
```

- ☐ C

```
class Three implements One, Two {
}
```

☒ D

```
class Three implements One, Two {
    public void method() {
        One.super.method();
    }
}
```

Q9. What is the output of this code?

```
class Main {
    public static void main (String[] args) {
        List list = new ArrayList();
        list.add("hello");
        list.add(2);
        System.out.print(list.get(0) instanceof Object);
        System.out.print(list.get(1) instanceof Integer);
    }
}
```

☐ The code does not compile.
☐ truefalse
☒ truetrue
☐ falsetrue

Q10. Given the following two classes, what will be the output of the Main class?

```
package mypackage;
public class Math {
    public static int abs(int num){
        return num < 0 ? -num : num;
    }
}
package mypackage.elementary;
public class Math {
    public static int abs (int num) {
        return -num;
    }
}

import mypackage.Math;
import mypackage.elementary.*;

class Main {
```

```
    public static void main (String args[]){
        System.out.println(Math.abs(123));
    }
}
```

- ☐ Lines 1 and 2 generate compiler errors due to class name conflicts.
- ☐ "-123"
- ☐ It will throw an exception on line 5.
- ☒ "123"

Explanation: The answer is "123". The `abs()` method evaluates to the one inside mypackage.Math class, because the import statements of the form:

`import packageName.subPackage.*`

is Type-Import-on-Demand Declarations, which never causes any other declaration to be shadowed.

Q11. What is the result of this code?

```
class MainClass {
    final String message() {
        return "Hello!";
    }
}

class Main extends MainClass {
    public static void main(String[] args) {
        System.out.println(message());
    }

     String message() {
         return "World!";
     }
 }
```

- ☒ It will not compile because of line 10.
- ☐ "Hello!"
- ☐ It will not compile because of line 2.
- ☐ "World!"

Explanation: Compilation error at line 10 because of final methods cannot be overridden, and here message() is a final method, and also note that Non-static method message() cannot be referenced from a static context.

Q12. Given this code, which command will output "2"?

```
class Main {
    public static void main(String[] args) {
```

```
        System.out.println(args[2]);
    }
}
```

- ☐ `java Main 1 2 "3 4" 5`
- ☒ `java Main 1 "2" "2" 5`
- ☐ `java Main.class 1 "2" 2 5`
- ☐ `java Main 1 "2" "3 4" 5`

Q13. What is the output of this code?

```
class Main {
    public static void main(String[] args){
        int a = 123451234512345;
        System.out.println(a);
    }
}
```

- ☐ "123451234512345"
- ☒ Nothing - this will not compile.
- ☐ a negative integer value
- ☐ "12345100000"

Reasoning: The int type in Java can be used to represent any whole number from -2147483648 to 2147483647. Therefore, this code will not compile as the number assigned to 'a' is larger than the int type can hold.

Q14. What is the output of this code?

```
class Main {
    public static void main (String[] args) {
        String message = "Hello world!";
        String newMessage = message.substring(6, 12)
            + message.substring(12, 6);
        System.out.println(newMessage);
    }
}
```

- ☐ The code does not compile.
- ☒ A runtime exception is thrown.
- ☐ "world!!world"
- ☐ "world!world!"

Q15. How do you write a for-each loop that will iterate over ArrayList<Pencil>pencilCase?

- ☒ `for (Pencil pencil : pencilCase) {}`
- ☐ `for (pencilCase.next()) {}`
- ☐ `for (Pencil pencil : pencilCase.iterator()) {}`

☐ `for (pencil in pencilCase) {}`

Q16. What does this code print?

```
System.out.print("apple".compareTo("banana"));
```

☐ 0
☐ positive number
☒ negative number
☐ compilation error

Q17. You have an ArrayList of names that you want to sort alphabetically. Which approach would NOT work?

☐ `names.sort(Comparator.comparing(String::toString))`
☐ `Collections.sort(names)`
☒ `names.sort(List.DESCENDING)`
☐ `names.stream().sorted((s1, s2) -> s1.compareTo(s2)).collect(Collectors.toList())`

Reference

Q18. By implementing encapsulation, you cannot directly access the class's __ properties unless you are writing code inside the class itself.

☒ private
☐ protected
☐ no-modifier
☐ public

Q19. Which is the most up-to-date way to instantiate the current date?

☐ `new SimpleDateFormat("yyyy-MM-dd").format(new Date())`
☐ `new Date(System.currentTimeMillis())`
☒ `LocalDate.now()`
☐ `Calendar.getInstance().getTime()`

Explanation: LocalDate is the newest class added in Java 8

Q20. Fill in the blank to create a piece of code that will tell whether `int0` is divisible by 5: `boolean isDivisibleBy5 = _____`

☐ `int0 / 5 ? true: false`
☒ `int0 % 5 == 0`
☐ `int0 % 5 != 5`
☐ `Math.isDivisible(int0, 5)`

Q21. How many times will this code print "Hello World!"?

```
class Main {
    public static void main(String[] args){
        for (int i=0; i<10; i=i++){
            i+=1;
            System.out.println("Hello World!");
        }
    }
}
```

- ☒ 10 times
- ☐ 9 times
- ☐ 5 times
- ☐ infinite number of times

Explanation: Observe the loop increment. It's not an increment, it's an assignment(post).

Q22. The runtime system starts your program by calling which function first?

- ☐ print
- ☐ iterative
- ☐ hello
- ☒ main

Q23. What code would you use in Constructor A to call Constructor B?

```
public class Jedi {
  /* Constructor A */
  Jedi(String name, String species){}

  /* Constructor B */
  Jedi(String name, String species, boolean followsTheDarkSide){}
}
```

- ☐ Jedi(name, species, false)
- ☐ new Jedi(name, species, false)
- ☒ this(name, species, false)
- ☐ super(name, species, false)

Note: This code won't compile, possibly a broken code sample.

Reference

Q24. "An anonymous class requires a zero-argument constructor." that's not true?

- ☐ An anonymous class may specify an abstract base class as its base type.
- ☒ An anonymous class does not require a zero-argument constructor.
- ☐ An anonymous class may specify an interface as its base type.
- ☐ An anonymous class may specify both an abstract class and interface as base types.

Q25. What will this program print out to the console when executed?

```
import java.util.LinkedList;

public class Main {
    public static void main(String[] args){
        LinkedList<Integer> list = new LinkedList<>();
        list.add(5);
        list.add(1);
        list.add(10);
        System.out.println(list);
    }
}
```

- ☒ [5, 1, 10]
- ☐ [10, 5, 1]
- ☐ [1, 5, 10]
- ☐ [10, 1, 5]

Q26. What is the output of this code?

```
class Main {
    public static void main(String[] args){
       String message = "Hello";
       for (int i = 0; i<message.length(); i++){
          System.out.print(message.charAt(i+1));
       }
    }
}
```

- ☐ "Hello"
- ☒ A runtime exception is thrown.
- ☐ The code does not compile.
- ☐ "ello"

Q27. Object-oriented programming is a style of programming where you organize your program around __ and data, rather than __ and logic.

- ☐ functions; actions
- ☒ objects; actions
- ☐ actions; functions

☐ actions; objects

Q28. What statement returns true if "nifty" is of type String?

☐ `"nifty".getType().equals("String")`
☐ `"nifty".getType() == String`
☐ `"nifty".getClass().getSimpleName() == "String"`
☒ `"nifty" instanceof String`

Q29. What is the output of this code?

```
import java.util.*;
class Main {
    public static void main(String[] args) {
        List<Boolean> list = new ArrayList<>();
        list.add(true);
        list.add(Boolean.parseBoolean("FalSe"));
        list.add(Boolean.TRUE);
        System.out.print(list.size());
        System.out.print(list.get(1) instanceof Boolean);
    }
}
```

☐ A runtime exception is thrown.
☐ 3false
☐ 2true
☒ 3true

Q30. What is the result of this code?

```
class Main {
    Object message() {
        return "Hello!";
    }
    public static void main(String[] args) {
        System.out.print(new Main().message());
        System.out.print(new Main2().message());
    }
}
class Main2 extends Main {
    String message() {
        return "World!";
    }
}
```

☐ It will not compile because of line 7.
☐ Hello!Hello!
☒ Hello!World!

☐ It will not compile because of line 11.

Q31. What method can be used to create a new instance of an object?

☐ another instance
☐ field
☒ constructor
☐ private method

Q32. Which is the most reliable expression for testing whether the values of two string variables are the same?

☐ string1 == string2
☐ string1 = string2
☐ string1.matches(string2)
☒ string1.equals(string2)

Q33. Which letters will print when this code is run?

```
public static void main(String[] args) {
    try {
        System.out.println("A");
        badMethod();
        System.out.println("B");
    } catch (Exception ex) {
        System.out.println("C");
    } finally {
        System.out.println("D");
    }
}
public static void badMethod() {
    throw new Error();
}
```

☐ A, B, and D
☐ A, C, and D
☐ C and D
☒ A and D

Explanation: `Error` is not inherited from `Exception`.

Q34. What is the output of this code?

```
class Main {
    static int count = 0;
    public static void main(String[] args) {
        if (count < 3) {
            count++;
```

```
            main(null);
        } else {
            return;
        }
        System.out.println("Hello World!");
    }
}
```

- ☐ It will throw a runtime exception.
- ☐ It will not compile.
- ☒ It will print "Hello World!" three times.
- ☐ It will run forever.

Q35. What is the output of this code?

```
import java.util.*;
class Main {
    public static void main(String[] args) {
        String[] array = {"abc", "2", "10", "0"};
        List<String> list = Arrays.asList(array);
        Collections.sort(list);
        System.out.println(Arrays.toString(array));
    }
}
```

- ☐ `[abc, 0, 2, 10]`
- ☐ The code does not compile.
- ☐ `[abc, 2, 10, 0]`
- ☒ `[0, 10, 2, abc]`

Explanation: The `java.util.Arrays.asList(T... a)` returns a fixed-size list backed by the specified array. (Changes to the returned list "write through" to the array.)

Q36. What is the output of this code?

```
class Main {
    public static void main(String[] args) {
        String message = "Hello";
        print(message);
        message += "World!";
        print(message);
    }
    static void print(String message) {
        System.out.print(message);
        message += " ";
    }
}
```

☐ Hello World!
☒ HelloHelloWorld!
☐ Hello Hello World!
☐ Hello HelloWorld!

Q37. What is displayed when this code is compiled and executed?

```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        x = 10;
        System.out.println(x);
    }
}
```

☐ x
☐ null
☒ 10
☐ 5

Q38. Which approach cannot be used to iterate over a List named *theList*?

☐ A

```
for (int i = 0; i < theList.size(); i++) {
    System.out.println(theList.get(i));
}
```

☐ B

```
for (Object object : theList) {
    System.out.println(object);
}
```

☒ C

```
Iterator it = theList.iterator();
for (it.hasNext()) {
    System.out.println(it.next());
}
```

☐ D

```
theList.forEach(System.out::println);
```

Explanation: `for (it.hasNext())` should be `while (it.hasNext())`.

Q39. What method signature will work with this code? `boolean healthyOrNot = isHealthy("avocado");`

- ☐ public void isHealthy(String avocado)
- ☒ boolean isHealthy(String string)
- ☐ public isHealthy("avocado")
- ☐ private String isHealthy(String food)

Q40. Which are valid keywords in a Java module descriptor (module-info.java)?

- ☐ provides, employs
- ☐ imports, exports
- ☐ consumes, supplies
- ☒ requires, exports

Q41. Which type of variable keeps a constant value once it is assigned?

- ☐ non-static
- ☐ static
- ☒ final
- ☐ private

Q42. How does the keyword `volatile` affect how a variable is handled?

- ☐ It will be read by only one thread at a time.
- ☐ It will be stored on the hard drive.
- ☒ It will never be cached by the CPU.
- ☐ It will be preferentially garbage collected.

Q43. What is the result of this code?

```
char smooch = 'x';
System.out.println((int) smooch);
```

- ☐ an alphanumeric character
- ☐ a negative number
- ☒ a positive number
- ☐ a ClassCastException

Q44. You get a NullPointerException. What is the most likely cause?

- ☐ A file that needs to be opened cannot be found.
- ☐ A network connection has been lost in the middle of communications.
- ☐ Your code has used up all available memory.
- ☒ The object you are using has not been instantiated.

Q45. How would you fix this code so that it compiles?

```java
public class Nosey {
    int age;
    public static void main(String[] args) {
        System.out.println("Your age is: " + age);
    }
}
```

- ☒ Make age static.
- ☐ Make age global.
- ☐ Make age public.
- ☐ Initialize age to a number.

Q46. Add a Duck called "Waddles" to the ArrayList ducks.

```java
public class Duck {
    private String name;
    Duck(String name) {}
}
```

- ☐ `Duck waddles = new Duck(); ducks.add(waddles);`
- ☐ `Duck duck = new Duck("Waddles"); ducks.add(waddles);`
- ☒ `ducks.add(new Duck("Waddles"));`
- ☐ `ducks.add(new Waddles());`

Q47. If you encounter `UnsupportedClassVersionError` it means the code was ___ on a newer version of Java than the JRE ___ it.

- ☐ executed; interpreting
- ☐ executed; compiling
- ☒ compiled; executing
- ☐ compiled, translating

Q48. Given this class, how would you make the code compile?

```java
public class TheClass {
    private final int x;
}
```

- ☐ A

```java
public TheClass() {
    x += 77;
}
```

- ☐ B

```java
public TheClass() {
    x = null;
}
```

☒ C

```
public TheClass() {
    x = 77;
}
```

☐ D

```
private void setX(int x) {
    this.x = x;
}
public TheClass() {
    setX(77);
}
```

Explanation: `final` class members are allowed to be assigned only in three places: declaration, constructor, or an instance-initializer block.

Q49. How many times f will be printed?

```
public class Solution {
    public static void main(String[] args) {
        for (int i = 44; i > 40; i--) {
            System.out.println("f");
        }
    }
}
```

☒ 4
☐ 3
☐ 5
☐ A Runtime exception will be thrown

Q50. Which statements about `abstract` classes are true?

```
1. They can be instantiated.
2. They allow member variables and methods to be inherited by subclasses.
3. They can contain constructors.
```

☐ 1, 2, and 3
☐ only 3
☒ 2 and 3
☐ only 2

Q51. Which keyword lets you call the constructor of a parent class?

☐ parent
☒ super
☐ this
☐ new

Q52. What is the result of this code?

```
1: int a = 1;
2: int b = 0;
3: int c = a/b;
4: System.out.println(c);
```

- ☒ It will throw an ArithmeticException.
- ☐ It will run and output 0.
- ☐ It will not compile because of line 3.
- ☐ It will run and output infinity.

Q53. Normally, to access a static member of a class such as Math.PI, you would need to specify the class "Math". What would be the best way to allow you to use simply "PI" in your code?

- ☒ Add a static import.
- ☐ Declare local copies of the constant in your code.
- ☐ This cannot be done. You must always qualify references to static members with the class from which they came from.
- ☐ Put the static members in an interface and inherit from that interface.

Q54. Which keyword lets you use an interface?

- ☐ extends
- ☒ implements
- ☐ inherits
- ☐ Import

Q55. Why are ArrayLists better than arrays?

- ☒ You don't have to decide the size of an ArrayList when you first make it.
- ☐ You can put more items into an ArrayList than into an array.
- ☐ ArrayLists can hold more kinds of objects than arrays.
- ☐ You don't have to decide the type of an ArrayList when you first make it.

Q56. Declare a variable that holds the first four digits of Π

- ☐ int pi = 3.141;
- ☐ decimal pi = 3.141;
- ☒ double pi = 3.141;
- ☐ float pi = 3.141;

Reasoning:

```
public class TestReal {
    public static void main (String[] argv)
    {
      double pi = 3.14159265;       //accuracy up to 15 digits
```

```
      float pi2 = 3.141F;           //accuracy up to 6-7 digits

      System.out.println ("Pi=" + pi);
      System.out.println ("Pi2=" + pi2);
    }
  }
```

```
The default Java type which Java will be used for a float variable will be double.
So, even if you declare any variable as float, what the compiler has to do is assign a doubl
which is not possible. So, to tell the compiler to treat this value as a float, that 'F' is
```

Q57. Use the magic power to cast a spell

```
public class MagicPower {
    void castSpell(String spell) {}
}
```

- ☒ `new MagicPower().castSpell("expecto patronum");`
- ☐ `MagicPower magicPower = new MagicPower(); magicPower.castSpell();`
- ☐ `MagicPower.castSpell("expelliarmus");`
- ☐ `new MagicPower.castSpell();`

Reference

Q58. What language construct serves as a blueprint containing an object's properties and functionality?

- ☐ constructor
- ☐ instance
- ☒ class
- ☐ method

Q59. What does this code print?

```
public static void main(String[] args) {
    int x=5,y=10;
    swapsies(x,y);
    System.out.println(x+" "+y);
}

static void swapsies(int a, int b) {
    int temp=a;
    a=b;
    b=temp;
}
```

- ☐ 10 10
- ☒ 5 10
- ☐ 10 5

☐ 5 5

Q60. What is the result of this code?

```
try {
    System.out.println("Hello World");
} catch (Exception e) {
    System.out.println("e");
} catch (ArithmeticException e) {
    System.out.println("e");
} finally {
    System.out.println("!");
}
```

☐ Hello World
☒ It will not compile because the second catch statement is unreachable
☐ Hello World!
☐ It will throw a runtime exception

Q61. Which is not a Java keyword

☐ finally
☐ native
☐ interface
☒ unsigned

Explanation: `native` is a part of the JNI interface.

Q62. Which operator would you use to find the remainder after division?

☒ `%`
☐ `//`
☐ `/`
☐ `DIV`

Reference

Q63. Which choice is a disadvantage of inheritance?

☐ Overridden methods of the parent class cannot be reused.
☐ Responsibilities are not evenly distributed between parent and child classes.
☒ Classes related by inheritance are tightly coupled to each other.
☐ The internal state of the parent class is accessible to its children.

Reference

Q64. How would you declare and initialize an array of 10 ints?

- ☐ `Array<Integer> numbers = new Array<Integer>(10);`
- ☐ `Array[int] numbers = new Array[int](10);`
- ☒ `int[] numbers = new int[10];`
- ☐ `int numbers[] = int[10];`

Q65. Refactor this event handler to a lambda expression:

```
groucyButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Press me one more time..");
    }
});
```

- ☐ `groucyButton.addActionListener(ActionListener listener -> System.out.println("Press me one more time..."));`
- ☒ `groucyButton.addActionListener((event) -> System.out.println("Press me one more time..."));`
- ☐ `groucyButton.addActionListener(new ActionListener(ActionEvent e) {() -> System.out.println("Press me one more time...");});`
- ☐ `groucyButton.addActionListener(() -> System.out.println("Press me one more time..."));`

Reference

Q66. Which functional interfaces does Java provide to serve as data types for lambda expressions?

- ☐ Observer, Observable
- ☐ Collector, Builder
- ☐ Filter, Map, Reduce
- ☒ Consumer, Predicate, Supplier

Reference

Q67. What is a valid use of the hashCode() method?

- ☐ encrypting user passwords
- ☒ deciding if two instances of a class are equal
- ☐ enabling HashMap to find matches faster
- ☐ moving objects from a List to a HashMap

Reference

Q68. What kind of relationship does "extends" denote?

- ☐ uses-a

☒ is-a
☐ has-a
☐ was-a

Reference

Q69. How do you force an object to be garbage collected?

☐ Set object to null and call Runtime.gc()
☒ Set object to null and call System.gc()
☐ Set object to null and call Runtime.getRuntime().runFinalization()
☐ There is no way to force an object to be garbage-collected

Reference

Q70. Java programmers commonly use design patterns. Some examples are the __, which helps create instances of a class, the __, which ensures that only one instance of a class can be created; and the __, which allows for a group of algorithms to be interchangeable.

☒ static factory method; singleton; strategy pattern
☐ strategy pattern; static factory method; singleton
☐ creation pattern; singleton; prototype pattern
☐ singleton; strategy pattern; static factory method

Q71. Using Java's Reflection API, you can use __ to get the name of a class and __ to retrieve an array of its methods.

☒ this.getClass().getSimpleName(); this.getClass().getDeclaredMethods()
☐ this.getName(); this.getMethods()
☐ Reflection.getName(this); Reflection.getMethods(this)
☐ Reflection.getClass(this).getName(); Reflection.getClass(this).getMethods()

Q72. Which is not a valid lambda expression?

☐ `a -> false;`
☐ `(a) -> false;`
☒ `String a -> false;`
☐ `(String a) -> false;`

Q73. Which access modifier makes variables and methods visible only in the class where they are declared?

☐ public
☐ protected
☐ nonmodifier
☒ private

Q74. What type of variable can be assigned only once?

- ☐ private
- ☐ non-static
- ☒ final
- ☐ static

Q75. How would you convert a String to an Int?

- ☐ `"21".intValue()`
- ☐ `String.toInt("21")`
- ☒ `Integer.parseInt("21")`
- ☐ `String.valueOf("21")`

Q76. What method should be added to the Duck class to print the name Moby?

```
public class Duck {
    private String name;

    Duck(String name) {
        this.name = name;
    }

    public static void main(String[] args) {
        System.out.println(new Duck("Moby"));
    }
}
```

- ☒ `public String toString() { return name; }`
- ☐ `public void println() { System.out.println(name); }`
- ☐ `String toString() { return this.name; }`
- ☐ `public void toString() { System.out.println(this.name); }`

Q77. Which operator is used to concatenate Strings in Java

- ☒ `+`
- ☐ `&`
- ☐ `.`
- ☐ `-`

Reference

Q78. How many times does this loop print "exterminate"?

```
for (int i = 44; i > 40; i--) {
    System.out.println("exterminate");
}
```

- ☐ two
- ☒ four
- ☐ three
- ☐ five

Q79. What is the value of myCharacter after line 3 is run?

```
public class Main {
  public static void main (String[] args) {
    char myCharacter = "piper".charAt(3);
  }
}
```

- ☐ p
- ☐ r
- ☒ e
- ☐ i

Q80. When should you use a static method?

- ☐ when your method is related to the object's characteristics
- ☒ when you want your method to be available independently of class instances
- ☐ when your method uses an object's instance variable
- ☐ when your method is dependent on the specific instance that calls it

Q81. What phrase indicates that a function receives a copy of each argument passed to it rather than a reference to the objects themselves?

- ☐ pass by reference
- ☐ pass by occurrence
- ☒ pass by value
- ☐ API call

Q82. In Java, what is the scope of a method's argument or parameter?

- ☒ inside the method
- ☐ both inside and outside the method
- ☐ neither inside nor outside the method
- ☐ outside the method

Q83. What is the output of this code?

```
public class Main {
  public static void main (String[] args) {
    int[] sampleNumbers = {8, 5, 3, 1};
    System.out.println(sampleNumbers[2]);
```

```
  }
}
```

- ☐ 5
- ☐ 8
- ☐ 1
- ☒ 3

Q84. Which change will make this code compile successfully?

```
public class Main {
  String MESSAGE ="Hello!";
  static void print(){
    System.out.println(message);
  }
  void print2(){}
}
```

- ☐ Change line 2 to `public static final String message`
- ☐ Change line 6 to `public void print2(){}`
- ☐ Remove the body of the `print2` method and add a semicolon.
- ☒ Remove the body of the `print` method.

Explanation: Changing line 2 to `public static final String message` raises the error `message not initialized in the default constructor`.

Q85. What is the output of this code?

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    String[] array = new String[]{"A", "B", "C"};
    List<String> list1 = Arrays.asList(array);
    List<String> list2 = new ArrayList<>(Arrays.asList(array));
    List<String> list3 = new ArrayList<>(Arrays.asList("A", new String("B"), "C"));
    System.out.print(list1.equals(list2));
    System.out.print(list1.equals(list3));
  }
}
```

- ☐ falsefalse
- ☒ truetrue
- ☐ falsetrue
- ☐ truefalse

Q86. Which code snippet is valid?

- ☐ `ArrayList<String> words = new ArrayList<String>(){"Hello", "World"};`

- ☐ `ArrayList words = Arrays.asList("Hello", "World");`
- ☐ `ArrayList<String> words = {"Hello", "World"};`
- ☒ `ArrayList<String> words = new ArrayList<>(Arrays.asList("Hello", "World"));`

Q87. What is the output of this code?

```
class Main {
  public static void main(String[] args) {
    StringBuilder sb = new StringBuilder("hello");
    sb.deleteCharAt(0).insert(0, "H").append(" World!");
    System.out.println(sb);
  }
}
```

- ☐ It will not compile.
- ☒ "Hello World!"
- ☐ "hello"
- ☐ ???? The code effectively converts the initial "hello" into "HelloWorld!" by deleting the first character, inserting "H" at the beginning, and appending " World!" to the end.

Q88. How would you use the TaxCalculator to determine the amount of tax on $50?

```
class TaxCalculator {
  static calculate(total) {
    return total * .05;
  }
}
```

- ☒ TaxCalculator.calculate(50);
- ☐ new TaxCalculator.calculate(50);
- ☐ calculate(50);
- ☐ new TaxCalculator.calculate($50);

Note: This code won't compile, broken code sample.

1. Reference
2. Code sample

Q89. Which characteristic does not apply to instances of java.util.HashSet?

- ☐ uses hashcode of objects when inserted
- ☐ contains unordred elements
- ☐ contains unique elements
- ☒ contains sorted elements

Explanation: HashSet makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time.

Reference

Q90. What is the output?

```
import java.util.*;

public class Main {
    public static void main(String[] args)
    {
        PriorityQueue<Integer> queue = new PriorityQueue<>();
        queue.add(4);
        queue.add(3);
        queue.add(2);
        queue.add(1);

        while (queue.isEmpty() == false) {
            System.out.printf("%d", queue.remove());
        }
    }
}
```

- ☐ 1 3 2 4
- ☐ 4 2 3 1
- ☒ 1 2 3 4
- ☐ 4 3 2 1

Q91. What will this code print, assuming it is inside the main method of a class? `System.out.println("hello my friends".split(" ")[0]);`

- ☐ my
- ☐ hellomyfriends
- ☒ hello
- ☐ friends

Q92. You have an instance of type Map<String, Integer> named instruments containing the following key-value pairs: guitar=1200, cello=3000, and drum=2000. If you add the new key-value pair cello=4500 to the Map using the put method, how many elements do you have in the Map when you call instruments.size()?

- ☐ 2
- ☐ When calling the put method, Java will throw an exception
- ☐ 4
- ☒ 3

Q93. Which class acts as the root class for the Java Exception hierarchy?

- ☐ Clonable
- ☒ Throwable
- ☐ Object
- ☐ Serializable

Q94. Which class does not implement the java.util.Collection interface?

- ☐ java.util.Vector
- ☐ java.util.ArrayList
- ☐ java.util.HashSet
- ☒ java.util.HashMap

Explanation: HashMap class implements Map interface.

Q95. You have a variable of named `employees` of type `List<Employee>` containing multiple entries. The `Employee` type has a method `getName()` that returns the employee name. Which statement properly extracts a list of employee names?

- ☐ `employees.collect(employee -> employee.getName());`
- ☐ `employees.filter(Employee::getName).collect(Collectors.toUnmodifiableList());`
- ☒ `employees.stream().map(Employee::getName).collect(Collectors.toList());`
- ☐ `employees.stream().collect((e) -> e.getName());`

Q96. This code does not compile. What needs to be changed so that it does?

```
public enum Direction {
    EAST("E"),
    WEST("W"),
    NORTH("N"),
    SOUTH("S");

    private final String shortCode;

    public String getShortCode() {
        return shortCode;
    }
}
```

- ☒ Add a constructor that accepts a `String` parameter and assigns it to the field `shortCode`.
- ☐ Remove the `final` keyword for the field `shortCode`.
- ☐ All enums need to be defined on a single line of code.

☐ Add a setter method for the field `shortCode`.

Q97. Which language feature ensures that objects implementing the `AutoCloseable` interface are closed when it completes?

☐ try-catch-finally
☐ try-finally-close
☒ try-with-resources
☐ try-catch-close

Q98. What code should go in line 3?

```
class Main {
    public static void main(String[] args) {
        array[0] = new int[]{1, 2, 3};
        array[1] = new int[]{4, 5, 6};
        array[2] = new int[]{7, 8, 9};
        for (int i = 0; i < 3; i++)
            System.out.print(array[i][1]); //prints 258
    }
 }
```

☐ `int[][] array = new int[][];`
☒ `int[][] array = new int[3][3];`
☐ `int[][] array = new int[2][2];`
☐ `int[][] array = [][];`

Q99. Is this an example of method overloading or overriding?

```
class Car {
    public void accelerate() {}
}
class Lambo extends Car {
    public void accelerate(int speedLimit) {}
    public void accelerate() {}
}
```

☐ neither
☒ both
☐ overloading
☐ overriding

Q100. Which choice is the best data type for working with money in Java?

☐ float
☐ String
☐ double

☒ BigDecimal

Reference

Q101. Which statement about constructors is not true?

☐ A class can have multiple constructors with a different parameter list.
☐ You can call another constructor with `this` or `super`.
☐ A constructor does not define a return value.
☒ Every class must explicitly define a constructor without parameters.

Q102. What language feature allows types to be parameters on classes, interfaces, and methods in order to reuse the same code for different data types?

☐ Regular Expressions
☐ Reflection
☒ Generics
☐ Concurrency

Q103. What will be printed?

```
public class Berries{

    String berry = "blue";

    public static void main(String[] args) {
        new Berries().juicy("straw");
    }
    void juicy(String berry){
        this.berry = "rasp";
        System.out.println(berry + "berry");
    }
}
```

☐ raspberry
☒ strawberry
☐ blueberry
☐ rasp

Q104. What is the value of `forestCount` after this code executes?

```
Map<String, Integer> forestSpecies = new HashMap<>();

forestSpecies.put("Amazon", 30000);
forestSpecies.put("Congo", 10000);
forestSpecies.put("Daintree", 15000);
forestSpecies.put("Amazon", 40000);
```

```
int forestCount = forestSpecies.size();
```

- ☒ 3
- ☐ 4
- ☐ 2
- ☐ When calling the put method, Java will throw an exception

Q105. What is the problem with this code?

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


class Main {

    public static void main(String[] args) {
        List<String> list = new ArrayList<String>(Arrays.asList("a", "b", "c"));
        for(String value :list) {
            if(value.equals("a")) {
                list.remove(value);
            }
        }
        System.out.println(list); // outputs [b,c]
    }
}
```

- ☐ String should be compared using == method instead of equals.
- ☒ Modifying a collection while iterating through it can throw a ConcurrentModificationException.
- ☐ The List interface does not allow an argument of type String to be passed to the remove method.
- ☐ ArrayList does not implement the List interface.

Q106. How do you convert this method into a lambda expression?

```
public int square(int x) {
    return x * x;
}
```

- ☐ `Function<Integer, Integer> squareLambda = (int x) -> { x * x };`
- ☐ `Function<Integer, Integer> squareLambda = () -> { return x * x };`
- ☒ `Function<Integer, Integer> squareLambda = x -> x * x;`
- ☐ `Function<Integer, Integer> squareLambda = x -> return x * x;`

Q107. Which choice is a valid implementation of this interface?

```
interface MyInterface {
    int foo(int x);
}
```

☐ A

```
public class MyClass implements MyInterface {
    // ....
    public void foo(int x){
        System.out.println(x);
    }
}
```

☐ B

```
public class MyClass implements MyInterface {
    // ....
    public double foo(int x){
        return x * 100;
    }
}
```

☒ C

```
public class MyClass implements MyInterface {
    // ....
    public int foo(int x){
        return x * 100;
    }
}
```

☐ D

```
public class MyClass implements MyInterface {
    // ....
    public int foo(){
        return 100;
    }
}
```

Q108. What is the result of this program?

```
interface Foo {
    int x = 10;
}

public class Main{
```

```
    public static void main(String[] args) {
        Foo.x = 20;
        System.out.println(Foo.x);
    }
}
```

- ☐ 10
- ☐ 20
- ☐ null
- ☒ An error will occur when compiling.

Q109. Which statement must be inserted on line 1 to print the value true?

```
1:
2: Optional<String> opt = Optional.of(val);
3: System.out.println(opt.isPresent());
```

- ☐ `Integer val = 15;`
- ☒ `String val = "Sam";`
- ☐ `String val = null;`
- ☐ `Optional<String> val = Optional.empty();`

Q110. What will this code print, assuming it is inside the main method of a class?

```
System.out.println(true && false || true);
System.out.println(false || false && true);
```

- ☐ false true
- ☐ true true
- ☒ true false
- ☐ false false

Q111. What will this code print?

```
List<String> list1 = new ArrayList<>();
list1.add("One");
list1.add("Two");
list1.add("Three");

List<String> list2 = new ArrayList<>();
list2.add("Two");

list1.remove(list2);
System.out.println(list1);
```

- ☐ `[Two]`
- ☒ `[One, Two, Three]`

☐ `[One, Three]`
☐ `Two`

Q112. Which code checks whether the characters in two Strings,named `time` and `money`, are the same?

☐ `if(time <> money){}`
☒ `if(time.equals(money)){}`
☐ `if(time == money){}`
☐ `if(time = money){}`

Q113. An __ is a serious issue thrown by the JVM that the JVM is unlikely to recover from. An __ is an unexpected event that an application may be able to deal with to continue execution.

☐ exception,assertion
☐ AbnormalException, AccidentalException
☒ error, exception
☐ exception, error

Q114. Which keyword would not be allowed here?

```
class Unicorn {
    _____ Unicorn(){}
}
```

☒ static
☐ protected
☐ public
☐ void

Q115. Which OOP concept is this code an example of?

```
List[] myLists = {
    new ArrayList<>(),
    new LinkedList<>(),
    new Stack<>(),
    new Vector<>(),
};

for (List list : myLists){
    list.clear();
}
```

☐ composition
☐ generics
☒ polymorphism
☐ encapsulation

Explanation: Switch between different implementations of the `List` interface.

Q116. What does this code print?

```
String a = "bikini";
String b = new String("bikini");
String c = new String("bikini");

System.out.println(a == b);
System.out.println(b == c);
```

- ☐ true; false
- ☒ false; false
- ☐ false; true
- ☐ true; true

Explanation: `==` `operator` compares the object reference. `String a = "bikini"; String b = "bikini";` would result in True. Here new creates a new object, so false. Use `equals() method` to compare the content.

Q117. What keyword is added to a method declaration to ensure that two threads do not simultaneously execute it on the same object instance?

- ☐ native
- ☐ volatile
- ☒ synchronized
- ☐ lock

Reference

Q118. Which is a valid type for this lambda function?

```
_____ oddOrEven = x -> {
    return x % 2 == 0 ? "even" : "odd";
};
```

- ☐ `Function<Integer, Boolean>`
- ☐ `Function<String>`
- ☒ `Function<Integer, String>`
- ☐ `Function<Integer>`

Explaination, Reference

Q119. What is displayed when this code is compiled and executed?

```
import java.util.HashMap;

public class Main {
```

```
    public static void main(String[] args) {
        HashMap<String, Integer> pantry = new HashMap<>();

        pantry.put("Apples", 3);
        pantry.put("Oranges", 2);

        int currentApples = pantry.get("Apples");
        pantry.put("Apples", currentApples + 4);

        System.out.println(pantry.get("Apples"));
    }
}
```

- ☐ 6
- ☐ 3
- ☐ 4
- ☒ 7

Explanation

Q120. What variable type should be declared for capitalization?

```
List<String> songTitles = Arrays.asList("humble", "element", "dna");
_______ capitalize = (str) -> str.toUpperCase();
songTitles.stream().map(capitalize).forEach(System.out::println);
```

- ☒ `Function<String, String>`
- ☐ `Stream<String>`
- ☐ `String<String, String>`
- ☐ `Map<String, String>`

Explanation, Reference

Q121. Which is the correct return type for the processFunction method?

```
_____ processFunction(Integer number, Function<Integer, String> lambda) {
    return lambda.apply(number);
}
```

- ☐ `Integer`
- ☒ `String`
- ☐ `Consumer`
- ☐ `Function<Integer, String>`

Explanation

Q122. What function could you use to replace slashes for dashes in a list of dates?

```
List<String> dates = new ArrayList<String>();
// missing code
dates.replaceAll(replaceSlashes);
```

- ☒ `UnaryOperator<String> replaceSlashes = date -> date.replace("/", "-");`
- ☐ `Function<String, String> replaceSlashes = dates -> dates.replace("-", "/");`
- ☐ `Map<String, String> replaceSlashes = dates.replace("/", "-");`
- ☐ `Consumer<Date> replaceSlashes = date -> date.replace("/", "-");`

Explanation: `replaceAll` method for any List only accepts UnaryOperator to pass every single element into it then put the result into the List again.

Q123. From which class do all other classes implicitly extend?

- ☒ `Object`
- ☐ `Main`
- ☐ `Java`
- ☐ `Class`

Explanation

Q124. How do you create and run a Thread for this class?

```
import java.util.date;

public class CurrentDateRunnable implements Runnable {
    @Override
    public void run () {
        while (true) {
            System.out.println("Current date: " + new Date());

            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

- ☒ `Thread thread = new Thread(new CurrentDateRunnable()); thread.start();`
- ☐ `new Thread(new CurrentDateRunnable()).join();`
- ☐ `new CurrentDateRunnable().run();`

☐ `new CurrentDateRunnable().start();`

Reference

Q125. Which expression is a functional equivalent?

```
List<Integer> numbers = List.of(1,2,3,4);
int total = 0;

for (Integer x : numbers) {
    if (x % 2 == 0)
    total += x * x;
}
```

☐ A

```
int total = numbers.stream()
                        .transform(x -> x * x)
                        .filter(x -> x % 2 == 0)
                        .sum ();
```

☐ B

```
int total = numbers.stream()
                        .filter(x -> x % 2 == 0)
                        .collect(Collectors.toInt());
```

☐ C

```
int total = numbers.stream()
                        .mapToInt (x -> {if (x % 2 == 0) return x * x;})
                        .sum();
```

☒ D

```
int total = numbers.stream()
                        .filter(x -> x % 2 == 0)
                        .mapToInt(x -> x * x)
                        .sum();
```

Explanation: The given code in the question will give you the output 20 as total:

```
numbers                         // Input `List<Integer>` > [1, 2, 3, 4]
    .stream()                   // Converts input into `Stream<Integer>`
    .filter(x -> x % 2 == 0)    // Filter even numbers and return `Stream<Integer>` > [2, 4
    .mapToInt(x -> x * x)       // Square the number, converts `Integer` to an `int`, and re
    .sum()                      // Returns the sum as `int` > 20
```

Q126. Which is not one of the standard input/output streams provided by java.lang.System?

- ☒ print
- ☐ out
- ☐ err
- ☐ in

Reference

Q127. The compiler is complaining about this assignment of the variable pickle to the variable jar. How would you fix this?

```
double pickle = 2;
int jar = pickle;
```

- ☐ Use the method toInt() to convert the pickle before assigning it to the jar.
- ☒ Cast pickle to an int before assigning it to the jar.
- ☐ Make pickle into a double by adding + ".0"
- ☐ Use the new keyword to create a new Integer from pickle before assigning it to the jar.

Reference

Q128. What value should x have to make this loop execute 10 times?

```
for(int i=0; i<30; i+=x) {}
```

- ☐ 10
- ☒ 3
- ☐ 1
- ☐ 0

Q129. The __ runs compiled Java code, while the __ compiles Java files.

- ☐ IDE; JRE
- ☐ JDK; IDE
- ☒ JRE; JDK
- ☐ JDK; JRE

Reference

Q130. Which packages are part of Java Standard Edition

- ☐ java.net
- ☐ java.util
- ☐ java.lang
- ☒ All above

Reference

Q131. What values for x and y will cause this code to print "btc"?

```
String buy = "bitcoin";
System.out.println(buy.substring(x, x+1) + buy.substring(y, y+2))
```

- ☒ int x = 0; int y = 2;
- ☐ int x = 1; int y = 3;
- ☐ int x = 0; int y = 3;
- ☐ int x = 1; int y = 3;

Q132. Which keyword would you add to make this method the entry point of the program?

```
public class Main {
    public static void main(String[] args) {
        // Your program logic here
    }
}
```

- ☐ exception
- ☐ args
- ☒ static
- ☐ String

Reference To make the main method the entry point of the program in Java, we need to use the static keyword. So, the correct answer is: static The main method must be declared as public static void main(String[] args) to serve as the entry point for a Java program

Q133. You have a list of Bunny objects that you want to sort by weight using Collections.sort. What modification would you make to the Bunny class?

```
//This is how the original bunny class looks
class Bunny{
    String name;
    int weight;

    Bunny(String name){
        this.name = name;
    }
    public static void main(String args[]){
        Bunny bunny = new Bunny("Bunny 1");
    }
}
```

- ☒ Implement the Comparable interface by overriding the compareTo method.
- ☐ Add the keyword default to the weight variable.

- ☐ Override the equals method inside the Bunny class.
- ☐ Implement Sortable and override the sortBy method.

Reference

Q134. Identify the incorrect Java feature.

- ☐ Object-oriented
- ☒ Use of pointers
- ☐ Dynamic
- ☐ Architectural neural

Reference

Q135. What is the output of this code?

```
int yearsMarried = 2;
switch (yearsMarried) {
   case 1:
      System.out.println("paper");
   case 2:
      System.out.println("cotton");
   case 3:
      System.out.println("leather");
   default:
      System.out.println("I don't gotta buy gifts for nobody!");
}
```

- ☐ cotton
- ☐ cotton leather
- ☒ cotton leather I don't gotta buy gifts for nobody!
- ☐ cotton I don't gotta buy gifts for nobody!

Reference

Q136. What language features do these expressions demonstrate?

```
System.out::println
Doggie::fetch
```

- ☐ condensed invocation
- ☐ static references
- ☒ method references
- ☐ bad code

Reference

Q137. What is the difference between the wait() and sleep() methods?

- ☐ Only Threads can wait, but any Object can be put to sleep.

- ☒ A waiter can be woken up by another Thread calling notification whereas a sleeper cannot.
- ☐ When things go wrong, sleep throws an IllegalMonitorStateException whereas wait throws an InterruptedException.
- ☐ Sleep allows for multi-threading whereas wait does not.

Reference

Q138. Which is the right way to declare an enumeration of cats?

- ☐ enum Cats (SPHYNX, SIAMESE, BENGAL);
- ☐ enum Cats (“sphynx”, “siamese”, “bengal”);
- ☒ enum Cats {SPHYNX, SIAMESE, BENGAL}
- ☐ enum Cats {“sphynx”,“siamese”,“bengal}

Q139. What happens when this code is run?

```
List<String> horses = new ArrayList<String>();
horses.add (" Sea Biscuit ");
System.out.println(horses.get(1).trim());
```

- ☐ “Sea Biscuit” will be printed.
- ☐ ” Sea Biscuit ” will be printed.
- ☒ An IndexOutOfBoundsException will be thrown.
- ☐ A NullPointerException will be thrown.

Q140. Which data structure would you choose to associate the amount of rainfall with each month?

- ☐ Vector
- ☐ LinkedList
- ☒ Map
- ☐ Queue

Explanation:

from @yktsang01 in #3915 thread

Map because the map is a key/value pair without creating new classes/objects. So can store the rainfall per month like `Map<java.time.Month, Double>`. The other options will most likely need some new class to be meaningful:

```
public class Rainfall {
    private java.time.Month month;
    private double rainfall;
}
Vector<Rainfall>
LinkedList<Rainfall>
Queue<Rainfall>
```

Q141. Among the following which contains date information?

- ☒ java.sql timestamp
- ☐ java.io time
- ☐ java.io.timestamp
- ☐ java.sql.time

Q142. What is the size of float and double in Java?

- ☒ 32 and 64
- ☐ 32 and 32
- ☐ 64 and 64
- ☐ 64 and 32

Q143. When you pass an object reference as an argument to a method call what gets passed?

- ☐ a reference to a copy
- ☒ a copy of the reference
- ☐ the object itself
- ☐ the original reference

Q144. Which choice demonstrates a valid way to create a reference to a static function of another class?

- ☒ Function<Integer, Integer> funcReference = MyClass::myFunction;
- ☐ Function<Integer, Integer> funcReference = MyClass()::myFunction();
- ☐ Function<Integer, Integer> funcReference = MyClass().myFunction;
- ☐ Function<Integer, Integer> funcReference = MyClass.myFunction();

Q145. What is UNICODE?

- ☐ Unicode is used for the external representation of words and strings
- ☐ Unicode is used for internal representation of characters and strings
- ☒ Unicode is used for external representation of characters and strings
- ☐ Unicode is used for the internal representation of words and strings

Q146. What kind of thread is the Garbage collector thread?

- ☐ User thread
- ☒ Daemon thread
- ☐ Both
- ☐ None of these

Q147. What is HashMap and Map?

- ☐ HashMap is Interface and map is a class that implements that
- ☐ HashMap is a class and map is an interface that implements that

☐ Map is a class and Hashmap is an interface that implements that
☒ Map is Interface and Hashmap is the class that implements that

Q148. What invokes a thread's run() method?

☐ JVM invokes the thread's run() method when the thread is initially executed.
☐ Main application running the thread.
☒ start() method of the thread class.
☐ None of the above.

Explanation: After a thread is started, via its `start()` method of the Thread class, the JVM invokes the thread's `run()` method when the thread is initially executed.

Q149. What is true about a final class?

☐ class declared final is a final class.
☐ Final classes are created so the methods implemented by that class cannot be overridden.
☐ It can't be inherited.
☒ All of the above.

Explanation: Final classes are created so the methods implemented by that class cannot be overridden. It can't be inherited. These classes are declared `final`.

Q150. Which method can be used to find the highest value of x and y?

☐ Math.largest(x,y)
☐ Math.maxNum(x,y)
☒ Math.max(x,y)
☐ Math.maximum(x,y)

Q151. `void accept(T t)` is method of which Java functional interface?

☒ Consumer
☐ Producer
☐ Both
☐ None

Q152. Which of these does `Stream filter()` operate on?

☒ Predicate
☐ Interface
☐ Class
☐ Methods

Q153. Which of these does `Stream map()` operates on?

- ☐ Class
- ☐ Interface
- ☐ Predicate
- ☒ Function

Q154. What code is needed at line 8?

```
1: class Main {
2:      public static void main(String[] args) {
3:          Map<String, Integer> map = new HashMap<>();
4:          map.put("a", 1);
5:          map.put("b", 2);
6:          map.put("c", 3);
7:          int result = 0;
8:
9:              result += entry.getValue();
10:         }
11:         System.out.println(result); // outputs 6
12:     }
13: }
```

- ☐ for(MapEntry<String, Integer> entry: map.entrySet()) {
- ☐ for(String entry: map) {
- ☐ for(Integer entry: map.values()) {
- ☒ for(Entry<String, Integer> entry: map.entrySet()) {

Q155. What will print when Lambo is instantiated?

```
class Car {
    String color = "blue";
}

class Lambo extends Car {
    String color = "white";

    public Lambo() {
        System.out.println(super.color);
        System.out.println(this.color);
        System.out.println(color);
    }
}
```

- ☒ blue white white
- ☐ blue white blue
- ☐ white white white
- ☐ white white blue

Q156. Which command will run a FrogSounds app that someone emailed to you as a jar?

- ☐ jar FrogSounds.java
- ☐ javac FrogSounds.exe
- ☐ jar cf FrogSounds.jar
- ☒ java -jar FrogSounds.jar

Q157. What is the default value of a short variable?

- ☒ 0
- ☐ 0.0
- ☐ null
- ☐ undefined

Q158. What will be the output of the following Java program?

```
class variable_scope {
    public static void main(String args[]) {
        int x;
        x = 5;
        {
            int y = 6;
            System.out.print(x + " " + y);
        }
        System.out.println(x + " " + y);
    }
}
```

- ☒ Compilation Error
- ☐ Runtime Error
- ☐ 5 6 5 6
- ☐ 5 6 5

Explanation: Scope of variable Y is limited.

Q159. Subclasses of an abstract class are created using the keyword __.

- ☒ extends
- ☐ abstracts
- ☐ interfaces
- ☐ implements

Reference

Q160. What will be the output of the following program?

```java
import java.util.Formatter;
public class Course {
    public static void main(String[] args) {
        Formatter data = new Formatter();
        data.format("course %s", "java ");
        System.out.println(data);
        data.format("tutorial %s", "Merit campus");
        System.out.println(data);
    }
}
```

- ☐ course java tutorial Merit campus
- ☒ course java course java tutorial Merit campus
- ☐ Compilation Error
- ☐ Runtime Error

Q161. Calculate the time complexity of the following program.

```java
void printUnorderedPairs(int[] arrayA, int[] arrayB){
    for(int i = 0; i < arrayA.length; i++){
        for(int  j = 0; j < arrayB.length; j++){
            if(arrayA[i] < arrayB[j]){
                System.out.println(arrayA[i] + "," + arrayB[j]);
            }
        }
    }
}
```

- ☐ O(N*N)
- ☐ O(1)
- ☒ O(AB)
- ☐ O(A*B)

Q162. What do these expressions evaluate?

```
1. true && false
2. true && false || true
```

- ☒ 1. false 2. true
- ☐ 1. false 2. false
- ☐ 1. true 2. false
- ☐ 1. true 2. true

Reference //check page number 47 and example number 4.:-}

Q163. What allows the programmer to destroy an object x?

- ☐ 1. x.delete()
- ☐ 2. x.finalize()

☐ 3. Runtime.getRuntime().gc()
☒ 4. Only the garbage collection system can destroy an object.

Reference //No, the Garbage Collection can not be forced explicitly. We may request JVM for garbage collection by calling System.gc() method. But This does not guarantee that JVM will perform the garbage collection

Q164. How many objects are eligible for garbage collection till flag

```
public class Test
{
    public static void main(String [] args)
    {
        Test obj1 = new Test();
        Test obj2 = m1(obj1);
        Test obj4 = new Test();
        obj2 = obj4;                //Flag
        doComplexStuff();
    }
    static Test m1(Test mx)
    {
        mx = new Test();
        return mx;
    }
}
```

☐ 1. 0
☒ 2. 1
☐ 3. 2
☐ 4. 4

Reference // question no 5.

Q165. Which interface definition allows this code to compile

```
int length = 5;
Square square = x -> x*x;
int a = square.calculate(length);
```

☐ A

```
@FunctionalInterface
public interface Square {
    void calculate(int x);
}
```

☒ B

```
@FunctionalInterface
public interface Square {
```

```
    int calculate(int x);
}
```

☐ C

```
@FunctionalInterface
public interface Square {
    int calculate(int... x);
}
```

☐ D

```
@FunctionalInterface
public interface Square {
    void calculate(int x, int y);
}
```

Reference

Q166. Which of the following represents the time complexity of an algorithm?

☐ O(N*N)
☐ O(1)
☐ O(A+B)
☒ O(A*B)

Reasoning: The answer option 'O(AB)' should be corrected to 'O(A*B)' to accurately represent the time complexity.

- O(N*N): This represents a quadratic time complexity, where the running time grows with the square of the input size.
- O(1): This represents constant time complexity, indicating that the algorithm's running time doesn't depend on the input size.
- O(A+B): This represents linear time complexity, indicating that the running time scales linearly with the sum of values A and B.
- O(A*B): This represents quadratic time complexity, indicating that the running time scales quadratically with the product of values A and B.

```
The original answer option 'O(AB)' is incorrect as it does not
properly represent a known time complexity notation. The correct
notation should be 'O(A*B)' to indicate quadratic time complexity.
```

Reference

Q167. Calculate the space complexity of the following program.

```
void createArray(int n) {
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = i * 2;
```

```
    }
}
```

- ☐ O(1)
- ☒ O(N)
- ☐ O(N^2)
- ☐ O(log(N))

//In this program, an array of size n is created. The space complexity is determined by the size of the dynamic array, which is n. Therefore, the space complexity is O(N).

Q167. What will be the output of the following Java code?

```
import java.util.*;
public class genericstack <E>
{
    Stack <E> stk = new Stack <E>();
public void push(E obj)
    {
        stk.push(obj);
}
public E pop()
    {
        E obj = stk.pop();
    return obj;
}
}
class Output
{
    public static void main(String args[])
    {
        genericstack <String> gs = new genericstack<String>();
        gs.push("Hello");
        System.out.println(gs.pop());
    }
}
```

- ☐ H
- ☒ Hello
- ☐ Runtime Error
- ☐ Compilation Error

//In this program, The code defines a generic stack class, pushes the string "Hello" onto the stack, and then pops and prints "Hello," resulting in the output "Hello."

Q168. In Java, what is the purpose of the synchronized keyword when used in the context of methods or code blocks?

- ☒ It is used to specify that a method or code block is asynchronous, allowing multiple threads to execute it concurrently.
- ☐ It is used to mark a method or code block as thread-safe, ensuring that only one thread can execute it at a time.
- ☐ It indicates that the method or code block is highly optimized for performance and will run faster than non-synchronized methods.
- ☐ It is used to prevent a method or code block from being executed by any thread, making it effectively "locked."

Q169. In Java, which of the following statements about the "transient" modifier is true?

- ☐ Transient variables cannot be accessed outside their declaring class.
- ☐ Transient variables are automatically initialized with a default value.
- ☒ Transient variables are not serialized when an object is serialized.
- ☐ Transient is a keyword used to define inner classes.

Q170. The following prototype shows that a Cylinder subclass is derived from a superclass called Circle.

- ☐ Class Circle extends Cylinder.
- ☐ Class Cylinder derived Circle.
- ☒ Class Cylinder extends Circle.
- ☐ Class Circle derived Cylinder.

Q171. What will be the output of the following Java code snippet?

```
class abc
{
    public static void main(String args[])
    {
        if(args.length>0)
        System.out.println(args.length);
    }
}
```

- ☒ The snippet compiles and runs but does not print anything.
- ☐ The snippet compiles, runs, and prints 0.
- ☐ The snippet compiles, runs, and prints 1.
- ☐ The snippet does not compile.

Q172. Which of these classes allows us to define our own formatting pattern for dates and times?

- ☐ DefinedDateFormat

☒ SimpleDateFormat
☐ ComplexDateFormat
☐ UsersDateFormatRead

Reference

Q173.What kind of relationship does extends denote?

☒ is-a
☐ has-a
☐ was-a
☐ uses-a