Preguntas Java Segunda Vuelta

1. ¿Para qué se usa el patrón de diseño builder?

   Se usa para simplificar la creación de objetos complejos

2. ¿Cuál es la diferencia entre throw y throws?

   throws indica el tipo de excepción que el método no va a manejar

   throw indica la excepción que arroja el método

3. Un método es declarado para aceptar 3 argumentos. Un programa llama este método y pasa solo dos argumentos. ¿Cuál es el resultado?

   Error de compilación

4. ¿Cuáles 3 implementaciones son válidas?

```
interface SampleCloseable {
     public void close() throws java.io.IOException; }
```

   A: class Test implements SampleCloseable{
   public void close() throws java.io.IOException {} }

   C: class Test implements SampleCloseable{
   Public void close() throws FileNotFoundException {} }

   E: class Test implements SampleCloseable{
   public void close() {} }

La segunda opción no puede ser porque Exception es padre de la excepción de la interfaz:

```
class Test implements SampleCloseable{
     public void close() throws Exception {} }
```

La cuarta opción no puede ser porque las clases no heredan de las interfaces, implementan:

```
class Test extends SampleCloseable{
     public void close() throws java.io.IOException {} }
```

5. ¿Cuáles 3 líneas compilarán e imprimirán “Right on!”?

```
public class Speak{
     public static void main(String[] args){
          Speak speakIT = new Tell();
          Tell tellIT = new Tell();
          speakIT.tellItLikeItIs();
          (Truth)speakIT.tellItLikeItIs();
          ((Truth)speakIT).tellItLikeItIs();
          tellIT.tellItLikeItIs();
          (Truth)tellIT.tellItLikeItIs();
          ((Truth)tellIT).tellItLikeItIs(); } }

Class Tell extends Speak implements Truth{
     @Override
     public void tellItLikeItIs(){
          System.out.println(“Right on!”); } }

Interface Truth{
     Public void tellItLikeItIs(); }
```

A: ((Truth)speakIT).tellItLikeItIs();

B: tellIT.tellItLikeItIs();

C: ((Truth)tellIT).tellItLikeItIs();

6. ¿Qué cambios harán que el código compile?

```
class X{
     X(){}
     private void one(){} }

public class Y extends X{
     Y(){}
     private void two(){
          one(); }
     public static void main(String[] args){
          new Y().two(); } }
```

Cambiar el modificador de acceso de one() de private a protected

7. ¿Qué imprime?

```
public class Main{
     public static void main(String[] args){
          int x = 2;
          for(; x<5;){
               x = x + 1;
               System.out.println(x); } } }
```

345

8. ¿Cuál es el resultado?

```
public class Test{
     public static void main(String[] args){
          int[][] array = {{0}, {0, 1}, {0, 2, 4}, {0, 3,
          6, 9}, {0, 4, 8, 12, 16}};
          System.out.println({4}{1});
```

```
System.out.println({1}{4}); } }
```

Error de compilación: la sintaxis no es correcta

9. ¿Cuáles son las dos posibles salidas?

```
public class Main throws Exception{
     public static void main(String[] args){
          doSomething(); }
     private static void soSomething() throws Exception{
          System.out.println(“Before if clause”);
          If(Math.random() > 0.5) throw new Exception();
          System.out.println(“After if clause”); } }
```

A: Before if clause Exception

B: Before if clause After if clause

10. ¿Cuál es la salida?

```
public class Main{
     public static void main(String[] args){
          int x = 2;
          if(x == 2) System.out.println(“A”);
          else System.out.println(“B”);
          else System.out.println(“C”); } }
```

Error de compilación: solo puede haber un else por cada if

11. ¿Cuántas veces se imprime el 2?

```
class Menu{
     public static void main(String[] args){
          String[] breakfast = {“beans”, “egg”, “ham”,
“juice”};
          for(String rs : breakfast){
               int dish = 1;
               while(dish < breakfast.length){
```

```
                    System.out.println(rs+“, ”+dish);
                    ++dish; } } } }
```

4: beans, 1 beans, 2 beans, 3 egg, 1 egg, 2 egg, 3 ham, 1 ham, 2 ham, 3 juice, 1 juice, 2 juice, 3

12. ¿Cuál es el resultado?

```
class Person{
     String name = “No name”;
     public Person (String nm) { name = nm; } }
class Employee extends Person{
     String empID = “0000”;
     public Employee (String id) { empID = id; } }
public class EmployeeTest{
     public static void main(String[] args){
          Employee e = new Employee(“4321”);
          System.out.println(e.empID); } }
```

Error de compilación: Se tiene que agregar un constructor sin parámetros ya que, al Employee heredar de Person, pasa por su constructor al crear un objeto de Employee

13. ¿Cuál es el resultado?

```
class Atom{
     Atom() { System.out.println(“atom ”); } }
class Rock extends Atom{
     Rock(String type) { System.out.println(type); } }
public class Mountain extends Rock{
     Mountain(){
          super(“granite ”);
          new Rock(“granite ”); }
     public static void main(String[] args){
          new Mountain(); } }
```

atom granite atom granite

Mountain hereda de Rock, que hereda de Atom, por lo que al crear un objeto Mountain primero pasa por el constructor vacío de Atom (imprime “atom ”), luego pasa por el constructor vacío de Rock y luego pasa por el contructor vacío de Mountain. La línea super(“granite ”) hace que pase por el constructor con un String de Rock (imprime “granite ”). Al crear un objeto Rock primero pasa por el constructor vacío de Atom (imprime “atom ”) y luego pasa por el constructor con un String de Rock (imprime “granite ”)

14. ¿Cuál es el resultado?

```
import java.text.*;
public class Align{
     public static void main(String[] args){
          String[] sa = {“111.234”, “222.5678”};
          NumberFormat nf = NumberFormat.getInstance();
          nf.setMaximumFractionDigits(3);
          for(String s : sa){
               System.out.println(nf.parse(s)); } } }
```

Error de compilacion: nf.parse(s) convierte la cadena en número pero no lo modifica. Sin embargo hay que poner un try catch al hacer esa conversión ya que la cadena podría no contener un número

15. ¿Cuál es el resultado?

```
interface Rideable{
     String getTicket(); }
public class Camel implements Rideable{
     int weight = 2;
     String getGait(){
          return mph + “, lope”; }
     void go(int speed){
          ++speed;
```

```
        weight++;
        int walkrate = speed * weight;
        System.out.println(walkrate + getGait()); }
    public static void main(String[] args){
        new Camel().go(8); } }
```

Error de compilación: no implementa el método de la interfaz

Suponiendo que el método de la interfaz es String getGait();, se debe poner public en la función de Camel para no reducir su visibilidad. Entra a la función go con 8. speed = 9. weight = 3 (¿porqué es una variable local?). walkrate = 27. Entra a la función getGait() y devuelve la variable mph (que no está declarada) más ", lope". Suponiendo que mph = "", se imprime "27, lope"

16. ¿Qué dos acciones, implementadas independientemente, permitirán a esta clase compilar?

```
import java.io.IOException;
public class Y{
    public static void main(String[] args){
        try{
            doSomething(); }
        catch(RuntimeException e){
            System.out.println(e); } }
    static void doSomething(){
        if(Math.random() > 0.5){
            throw new IOException(); }
        throw new RuntimeException(); } }
```

A: Agregar throws IOException al main y al doSomething

B: Agregar throws IOException al doSomething y cambiar el catch por IOException

17. ¿Cuál es el resultado?

```
try{
      //conn es un objeto Connection
      //un objeto Statement es creado
      //las llamadas rollback serán válidas
      //se usa SQL para agregar 10 a una cuenta de cheques
      Savepoint s1 = conn.setSavePoint();
      //se usa SQL para agregar 100 a la misma cuenta
      Savepoint s2 = conn.setSavePoint();
      //se usa SQL para agregar 1000 a la misma cuenta
      //insertar aquí una invocación válida del método rollback
      } catch(Exception e) {}
```

18. ¿Cuáles de las siguientes opciones son válidas?

A: El constructor predeterminado puede ser llamado usando this();

B: Un constructor puede ser invocado desde un método de instancia

C: Una variable de instancia puede ser accedida dentro de un método de clase

D: Un constructor puede ser llamado dentro de otro constructor usando la palabra clave this()

19. ¿Cuál es el resultado?

```
class X{
      static void m(int i){
            i += 7; }
      public static void main(String[] args){
            int i = 12;
            m(i);
            System.out.println(i); } }
```

12: la función m no retorna el nuevo valor de i

20. ¿Cuál es el resultado si el valor de value es 33?

```
public static void main(String[] args){
     if(value >= 0){ //true
          if(value != 0) //true -> the
               System.out.println("the ");
          else
               System.out.println("quick ");
          if(value < 10) //false
               System.out.println("brown ");
          if(value > 30) //true -> the fox
               System.out.println("fox ");
          else if(value < 50)
               System.out.println("jumps ");
          else if(value < 10)
               System.out.println("over ");
          else
               System.out.println("the ");
          if(value > 10) //true -> the fox lazy
               System.out.println("lazy ");
          else
               System.out.println("dog ");
          System.out.println("…"); } } //the fox lazy…
```

the fox lazy…

21. ¿Cuál es el resultado si intentas compilar Truthy.java y luego lo ejecutas con las assertions habilitadas?

```
public class Truthy{
     public static void main(String[] args){
          int x = 7;
          assert (x == 6) ? "x == 6" : "x != 6"; } }
```

Error de compilación

assert tiene que regresar un boolean, no una cadena. Para ejecutarlo hay que inicializarlo

22. Definición del Singleton

El patrón de diseño Singleton se caracteriza por tener solo una instancia pero con acceso global

23. Diferencias entre clases abstractas e interfaces

A: Una clase puede heredar de una interfaz

B: Una clase puede heredar de múltiples clases

C: En una interfaz puede haber métodos con y sin comportamiento

Una clase puede implementar múltiples interfaces. En una clase abstracta puede haber métodos con comportamiento

Una clase abstract puede tener constructor

En las interfaces los métodos son public