

Desarrollo de Software

Apellidos y Nombres: Lezama Verástegui Dagmar Nicole

Código: 20204096K

Prueba de entrada

1. Coloca tus nombres completos en el documento donde están tus respuestas.
2. Debes escribir código para explicar tus respuestas, no se admite solo texto.
3. En esta tarea se evaluará todas las preguntas y no se asignará puntaje a preguntas incompletas.
4. Incluir imágenes o formas de verificar la ejecución de los programas planteados.
5. Todo acto de COPIA implica la nota de 0A. ¡Evita copiar!.

Entrega de la prueba

En la plataforma de Google Classroom deberás entregar la dirección de github personal creado en la actividad 1, donde se debe encontrar un repositorio llamado Curso-CC3S2 y dentro de él debes crear un directorio llamado **PracticaEntrada** donde se encuentran todos tus archivos de respuesta, escritos de manera ordenada.

Preguntas

En todos los casos, debes escribir programas funcionales que se pueden ejecutar utilizando IntelliJ IDEA

1 ¿Cuál es la salida del siguiente programa? (3 puntos)

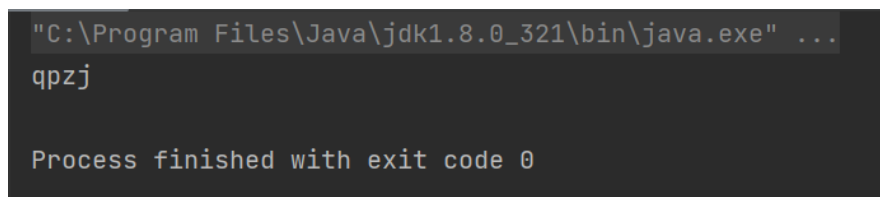
```
1: class Canine {
2: public Canine(boolean t) { logger.append("a"); }
3: public Canine() { logger.append("q"); }
4:
5: private StringBuilder logger = new StringBuilder();
6: protected void print(String v) { logger.append(v); }
7: protected String view() { return logger.toString(); }
8: }
9:
10: class Fox extends Canine {
11: public Fox(long x) { print("p"); }
12: public Fox(String name) {
13: this(2);
14: print("z");
15: }
16: }
17:
18: public class Fennec extends Fox {
19: public Fennec(int e) {
20: super("tails");
```

```

21: print("j");
22: }
23: public Fennec(short f) {
24: super("eevee");
25: print("m");
26: }
27:
28: public static void main(String... unused) {
29: System.out.println(new Fennec(1).view());
30: }
31: }

```

A continuación, se presenta una captura de la ejecución asociada al problema.



```

"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
qpzj

Process finished with exit code 0

```

Podemos apreciar que lo que se muestra es qpzj, esto se debe a que los constructores existentes se ejecutan de forma descendente, dándose prioridad a las clases padres mediante los constructores `super()` o `this()` de las clases hijas.

Lo que sucede en el método `main()` es que se imprime la secuencia mostrada tras primero hacer el llamado al constructor `Fennec()` y este al recibir un entero, hace que se llame a uno de los constructores de `Fox()`, aquel que tiene como parámetro un `String` e inmediatamente se llama al otro constructor `Fox()` que recibe un entero gracias a `this(2)`; a pesar de no haber una llamada a un constructor padre, se inserta una llamada al constructor sin argumento `super()` ya que esta clase extiende a la clase `Canino()` y se usa el constructor que no tiene ningún argumento. De este modo, se obtiene la secuencia qpzj.

2 ¿Qué afirmación sobre el siguiente programa es correcta? Si encuentras un error (es) corrígelo y presenta una respuesta correcta del código (3 puntos)

```

1: class Bird {
2: int feathers = 0;
3: Bird(int x) { this.feathers = x; }
4: Bird fly() {
5: return new Bird(1);
6: } }
7: class Parrot extends Bird {
8: protected Parrot(int y) { super(y); }
9: protected Parrot fly() {
10: return new Parrot(2);
11: } }
12: public class Macaw extends Parrot {
13: public Macaw(int z) { super(z); }
14: public Macaw fly() {
15: return new Macaw(3);

```

```
16: }
17: public static void main(String... sing) {
18: Bird p = new Macaw(4);
19: System.out.print(((Parrot)p.fly()).feathers);
20: }}
```

- a) Una línea contiene un error del compilador
- b) Dos líneas contienen errores del compilador
- c) Tres líneas contienen errores del compilador
- d) El código se compila pero lanza una ClassCastException en tiempo de ejecución
- e) El programa compila e imprime 3.
- f) El programa compila e imprime 0.

Al momento de compilarlo tal y como se presenta, nos muestra el siguiente error:

```
C:\Users\Nicole\Documents\2022-2\PruebaddeEntrada\src\P2\Main.java:35:22
java: non-static variable this cannot be referenced from a static context
```

Para solucionar esta problemática es necesario que nuestras clases sean estáticas también

```
static class Bird
static class Parrot extends Bird
static public class Macaw extends Parrot
```

Así el programa compila e imprime 3, como se puede apreciar:

```
"C:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
3
Process finished with exit code 0
```

Podemos apreciar que existe una buena implementación de los constructores de cada clase, así como un buen manejo de la sobreescritura del método fly().

En el main se crea un objeto Bird p que es del tipo Macaw, lo cual es correcto porque Macaw extiende a Parrot que también extiende a Bird, podemos fijarnos que en la sobreescritura del método fly() en la clase Macaw, a feathers se le asigna al valor de 3. Y en lo que se imprime, vemos que p se convierte en clase Parrot, pero como se accede a lo iniciado antes se imprime el valor de feathers que es 3.

3 ¿Cuáles de las siguientes afirmaciones sobre la herencia son correctas? Explica tu respuesta (2 puntos)

- a) Una clase puede extender directamente cualquier número de clases.
- b) Una clase puede implementar cualquier número de interfaces.
- c) Todas las variables heredan java.lang.Object.
- d) Si la clase A se extiende por B, entonces B es una superclase de A.
- e) Si la clase C implementa la interfaz D, entonces C es un subtipo de D.
- f) La herencia múltiple es la propiedad de una clase de tener múltiples superclases directas.

Las afirmaciones correctas son las que pertenecen a las claves a) b) e) y f)

- a) Esta opción es verdadera, pues a pesar de que en Java una clase solo puede extender directamente a una única clase, en la herencia respecto a POO existe el concepto de herencia múltiple que concuerda con este enunciado.
- b) Es verdadero, ya que una clase si tiene la posibilidad de implementar cualquier número de interfaces como desee.
- c) Esta opción es falsa, ya que esto no ocurre para las variables de tipo primitivo.
- d) Falso, ya que cuando una clase extiende a otra clase, se convierte subclase de aquella clase, o sea que B sería subclase de A.
- e) Verdadero, ya que una clase que implementa una interfaz se denomina subtipo de esa interfaz.
- f) Verdadero, ya que brinda una descripción que concuerda con la herencia múltiple, a pesar de esta no ser admitida en Java

4 Completa los espacios en blanco que permiten que se compile el código. (4 puntos)

```

1: public class Howler {
2: public Howler(long shadow) {
3: _____;
4: }
5: private Howler(int moon) {
6: super();
7: }
8: }
9: class Wolf extends Howler {
10: protected Wolf(String stars) {
11: super(2L);
12: }
13: public Wolf() {
14: _____;
15: }
16: }

```

Básicamente se trata de completar con `this()` que contengan un parámetro coincidente con algún constructor existente que se relacione con las clases respectivas.

En la caso Howler, al momento de insertar un `this()` que tenga un parámetro entero, se llamará al constructor de `Howler(int moon)` pues coincide con el parámetro requerido de tipo `int`. Y al agregar `this("guau")`, se llamará al constructor que admite un `String` y no habrá problemas en la compilación.

5 ¿Cuál es el resultado del siguiente código? (3 puntos)

```

1: interface Climb {
2: boolean isTooHigh(int height, int limit);
3: }
4:
5: public class Climber {
6: public static void main(String[] args) {
7: check((h, m) -> h.append(m).isEmpty(), 5);

```

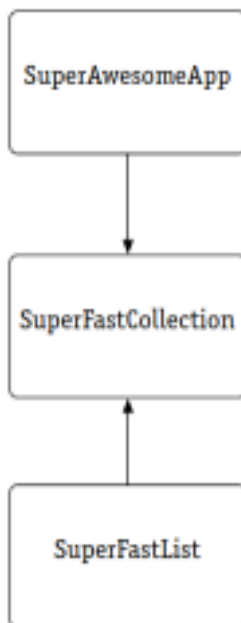
```

8: }
9: private static void check(Climb climb, int height) {
10: if (climb.isTooHigh(height, 10))
11: System.out.println("muy alto");
12: else
13: System.out.println("ok");
14: }
15: }

```

6 Responde las siguientes preguntas (5 puntos)

Jessica Abejita, está diseñando una aplicación SuperAwesomeApp que utiliza un tipo de datos que inventó llamado SuperFastList. Revisó las notas de la clase CC-3S2 y se dio cuenta de que debía **desacoplar** la aplicación del tipo de datos con una interfaz, por lo que ideó el siguiente diagrama de dependencia, en el que SuperFastCollection es una interfaz. Desafortunadamente, se olvidó de su idea inicial y necesita un consejo, así que los llamo a ustedes que asisten a clases.



¿Qué puedes decir de las siguientes declaraciones ?

- a) El diagrama también debe mostrar una dependencia de SuperAwesomeApp en SuperFastList, ya que una clase no puede depender solo de una interfaz y debe tener acceso a la implementación.
- b) SuperFastList depende de SuperFastCollection porque lo implementa, y un cambio en la interfaz probablemente implica un cambio en la clase.
- c) Si este diagrama es correcto, no se puede crear una instancia de SuperFastList dentro de SuperAwesomeApp, pero se puede pasar un objeto de tipo SuperFastList a SuperAwesomeApp.

Jessica les dice que su amigo CesarL. Hacker señaló que SuperFastList implementa correctamente todos los métodos declarados en la interfaz `java.util.List`. Él le sugirió que debería reemplazar la interfaz SuperFastCollection en el diagrama por `java.util.List`.

Jessica quiere tu opinión.

- a) Sería deseable reemplazar SuperFastCollection por java.util.List si es posible
- b) No será posible reemplazar SuperFastCollection por java.util.List si SuperFastCollection contiene métodos adicionales que no están en java.util.List
- c) Puede que no sea posible reemplazar SuperFastCollection por java.util.List, porque SuperAwesomeApp puede llamar a métodos que no pertenecen a java.util.List
- d) Puede que no sea posible reemplazar SuperFastCollection por java.util.List, porque SuperFastList puede contener métodos que no pertenecen a java.util.List

Jessica descubre que su aplicación necesita un método de lista extra para conquerWorld que actualmente no es un método de SuperFastList. Entonces crea una subclase ExtraSuperFastList de SuperFastList. Actualmente, SuperAwesomeApp usa solo una lista superrápida que se crea fuera y se pasa como parámetro. CesarL advierte a Jessica que es posible que deba cambiar la forma en que se pasa este parámetro.

Para que SuperAwesomeApp llame a este nuevo método, ¿cuál debe ser el tipo declarado/tipo de tiempo de ejecución del parámetro?

- a) SuperFastCollection / SuperFastList
- b) SuperFastCollection / ExtraSuperFastList
- c) SuperFastList / ExtraSuperFastList
- d) ExtraSuperFastList / ExtraSuperFastList

Nota: debes presentar imágenes de la ejecución y los códigos corregidos sobre IntelliJ IDEA.