Taller seis. Herencia y constructores

Programación orientada objetos – 2021-1S

Simón Cuartas Rendón – C.C. 1.037.670.103
Estudiante de estadística

Ejercicio uno.

1. ¿Qué imprime el código dado cuando se crea una nueva instancia de `Cuarto`?
   Al crear una nueva instancia de `Cuarto` en una nueva clase asociada a las dadas, se obtiene el siguiente código:

```
1  package parteUno;
2
3  public class ejecucionUo {
4      public static void main(String[] args) {
5          Cuarto instancia = new Cuarto();
6      }
7
8  }
```

```
<terminated> ejecucionUo [
m desde Cuarto
m desde Segundo
```

   Entonces, para obtener estas dos líneas, ocurrió la siguiente secuencia:

   a. Al crearse una instancia (en este caso llamada `instancia`) de `Cuarto`, este busca su constructor, pero como no existe, entonces el compilador lo agrega automáticamente, esto es, está adicionando un constructor por defecto, el cual va a tener como primera y única línea un `super()` pues se tiene que la clase `Cuarto` es subclase de `Tercero`.

```
class Cuarto extends Tercero {

    Cuarto() { // This constructor is added by the compiler.
        super();
    }
```

   b. Con este llamado del constructor por defecto de `Cuarto`, se llega a `Tercero`, el cual a pesar de poseer un constructor definido explícitamente, este carece del llamada al constructor de su clase padre, de modo que el compilador lo agrega automáticamente en la primera línea de `Tercero()`.

```
class Tercero extends Segundo {
    Tercero() {
        super(); // This lines is added by the compiler.
        super.m();
    }
```

c. Nuevamente, se dirige al constructor de su clase padre, en este caso, de **Segundo**, el cual también carece de un constructor definido explícitamente, por lo que sucede un proceso análogo al descrito en el literal (a).

```
class Segundo extends Primero {

    Segundo() { // This constructor is added by the compiler.
        super();
    }
```

d. Y con esto, se llama al constructor de la clase padre de **Segundo**, el cual es **Primero** y que tiene un constructor bien definido. Aquí, como no hay ninguna clase padre definida explícitamente para **Primero**, es decir, solo es hija de la clase **Object**, entonces no se sigue 'ascendiendo' y se queda en este constructor, el cual ejecuta el método **m(),** y en este caso, por ligadura dinámica, ejecuta el método **m()** asociado a clase **Cuarto**, con lo cual se obtiene la primera línea de la consola, es decir, *"m desde Cuarto."*

```
Primero() {
    m(); // Executes the method in Cuarto by dynamic link.
}
```

e. Ya 'cerrado' este constructor, se va a dirigir al constructor de su clase hija, que es **Segundo.**
f. En **Segundo** también se 'cierra' el constructor y se regresa al de su clase hija, **Tercero.**
g. Estando en **Tercero**, al ya haber ejecutado la primera línea de su constructor, toma la segunda, **super.m(),** la cual le está indicando que ejecute el método **m()** de la superclase, en este caso, de **Segundo**, la cual imprime por consola la segunda línea: *"m desde Segundo."*
h. Habiendo realizado todas las indicaciones dadas por el constructor de **Tercero**, puede regresar al constructor de **Cuarto** donde ya no hay más instrucciones, por lo que finaliza el ciclo.

Ejercicio dos.

1. Identificar los errores del código dado y explicar cómo se pueden resolver.

1.1. `Estudiante e1 = new Estudiante(91, "Sara", 1431);`

Este error aparece porque no se identifica ningún constructor que pueda asignar los atributos como se indican en esa línea de código, y el constructor de la clase padre que se ejecuta a través del constructor por defecto agregado sería inservible porque este solo acepta dos parámetros, cuando aquí tenemos tres. Así, bastaría con agregar un constructor que funcione como se pretende en esa línea:

```
Estudiante(int codigo, String nombre, long cedula) {
    super(nombre, cedula);
    this.codigo = codigo;
```

1.2. `Persona p2 = new Estudiante(87, "Sofia", "4");`

Habiendo ya un constructor para los objetos de la clase **Estudiante,** se debe notar que este recibe un número entero, un **String** y otro número entero de tipo **long,** pero aquí se pretende pasar como cuarto parámetro un **String**, por lo que se pueden plantear dos soluciones:

a. Bajo la lógica de que la cédula se trata como un entero de tipo `long`, bastaría con eliminar las comillas que rodean al número cuatro.
```
Persona p2 = new Estudiante(87, "Sofia", 4);
```
b. Si se quiere dejar la opción de que el usuario ingrese la cédula como un `String`, se puede definir un método que lo reciba de la misma forma que luego realice un proceso de conversión para podérselo asignar al atributo correspondiente.
```
Estudiante(int codigo, String nombre, String cedula) {
        super(nombre, Long.parseLong(cedula));
        this.codigo = codigo;
```

1.3. `super("Juan", 20123);`

Justo encima de esta línea se está llamando a un `this`, lo cual genera un error en tanto ambos deben estar en la primera línea de los constructores siempre, por lo que no es posible usarlos simultáneamente. Con esto, solo se debe borrar a uno de los tres para resolver la situación, y podría ser eliminando el `super`, pues con el `this` se está llamando al otro constructor de `Profesor` con el cual se hace el mismo procedimiento que se quería lograr.
```
// super("Juan", 20123); <- This line must be commented!
```

1.4. `super.nombre += n;`

Aquí, se tiene que `nombre` no es un atributo propio de esta clase, sino que es heredado de `Persona`, y el inconveniente surge porque el nombre está privado, lo que se puede solucionar flexibilizando la visibilidad de este atributo. Una buena opción, en aras de la mayor privacidad posible, podría ser dejarlo de tipo `paquete` o por defecto.
```
String nombre;
```

1.5. `public final int getEdad() {`

Con esta línea el problema resulta en que la clase `Persona`, padre de `Estudiante`, también tiene definido un método `getEdad()`, con el agravante de que tiene la palabra clave `final`, lo cual expresa que este método no puede ser heredado ni sobrescrito, lo que quiere decir que al simplemente borrar tal palabra se puede resolver el problema.
```
public int getEdad() { // This is at Persona class.
```

Hecho esto, ya será posible compilar y ejecutar el método.