Constructores:

Ejercicios:

1. Salida:

Pruebas 1

123

123

Lo primero que se registra en la salida es un “Pruebas 1”, en forma de una línea, al ser un System.out.println la primera línea de código del main de la clase PruebaConstructores. Luego le siguen dos líneas con “123”, esto porque se hace primero una asignación de una Clase3 a una variable de tipo Clase3, lo que dispara las llamadas al constructor de la clase. Pero como Clase3 hereda de Clase2 y Clase2 de Clase1, el primer constructor en ser llamado es el de Clase1 (que tiene un print(“1”)), luego Clase2(que tiene un print(“2”)) y finalmente Clase3 (que tiene un println(“3”)). La segunda asignación del main produce la misma respuesta, solo que se asigna una Clase3 a una variable de tipo Clase2, algo correcto ya que Clase3 es una Clase2.

1. Salida:

La salida dará una excepción, ya que el constructor de ClaseC sin parámetros no está definido y ClaseA tiene su constructor como private.

Ejercicios, parte 2:

Salida:

123

Se registrará en la salida una única línea que diría “123”. Esto se debe a que en el main de la clase Ctest hay una única asignación de una clase C3 a una variable C3, lo que dispara la llamada al constructor de la clase. Pero como C3 hereda de C2 y C2 de C1, el primer constructor en ser llamado es el de C1 (que tiene un print(“1”)), luego C2(que tiene un print(“2”)) y finalmente C3 (que tiene un print(“3”)).

Cuestionario:

1. Es necesario invocar explícitamente al constructor de la clase base cuando se le pasan parámetros a la subclase (durante la asignación) que corresponden a atributos de la clase base.
2. No es necesario definir un constructor en todas las clases, si no está definido el compilador agregará al constructor por defecto en tiempo de compilación, y también seteará los valores de los atributos (si los tuviera) a los valores nulos de sus tipos.
3. No, es igual que en una clase común, puede o no tenerlo.
4. Una interfaz no puede tener constructor, ya que no genera un objeto por si sola, por lo tanto no hay necesidad de tener un constructor.
5. En una jerarquía de clases los constructores se ejecutan desde el primer objeto padre hasta el último hijo.
6. Si la clase padre tiene un constructor por defecto el compilador agregará super() como primera línea del constructor de la clase hija (en tiempo de compilación), pero si no lo hubiera dará error y habrá que declarar explícitamente la llamada al correspondiente constructor de la clase padre.
7. Verdadero, los constructores son propios de cada clase.

Modificadores de visibilidad:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Modificadores de los atributos | | | |
| Visibilidad | private | package | protected | public |
| Misma clase | Si | Si | Si | Si |
| Mismo paquete | No | Si | Si | Si |
| Subclase en mismo paquete | No | Si | Si | Si |
| Subclase en otro paquete | No | No | Si | Si |
| Cualquier clase que no sea subclase en otro paquete | No | No | No | Si |

Colecciones:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ArrayList | LinkedList | TreeSet | HashMap |
| Agregar | Si | Si | Si | Si |
| Eliminar | Si | Si | Si | Si |
| Recorrido | Si, y tiene iterador | Si, y tiene iterador | Si, y tiene iterador | Si |
| Orden de los elementos | Si | Si | Si | No se garantiza ningún orden. |
| Escenario de uso ideal | Cuando no se conoce  la cantidad de objetos  que se desea guardar | Para crear pilas, colas, árboles. | Para almacenar datos ordenados. | Para guardar datos que tengan asociado un determinado valor. Por ejemplo: un id de producto y su precio. |
| Admite duplicados | Si | Si | No | No admite claves duplicadas, pero sí pueden repetirse los valores para esas claves. |

Excepciones:

1. La diferencia es que en la excepción aritmética esta lanzó una excepción del tipo ArithmeticException, en cambio en la excepción input-output lanzó una del tipo FileNotFoundException. Lo que indica que las excepciones aritméticas están englobadas dentro de una sola, pero las IO están diferenciadas según qué las produce. Además, la IOException dió un mensaje de error (El sistema no puede encontrar el archivo especificado) mucho más claro que el de la ArithmeticException (/ by zero).
2. El bloque finally sirve para evitar lo que es conocido como fuga de recursos, en Java esto puede suceder cuando no se cierran apropiadamente archivos, conexiones a bases de datos o conexiones de red. Finally es utilizado para tal caso, siendo además opcional.
3. La prioridad se da por el orden de ejecución, primero el try, segundo el catch y tercero el finally.
4. Si hubiera un return dentro del bloque cath y también existiera un finally, se ejecutarán ambos. Pero antes que el return se ejecutará el finally.

Static:

1. Una importación estática es la importación de una clase para utilizar sus métodos estáticos sin tener que hacer referencias continuamente a la clase.
2. Un atributo static es un atributo de la clase, por lo tanto, no necesita de una instancia para existir.
3. Un método static al igual que un atributo static es un método de la clase, por lo tanto, puede ser usado sin la necesidad de una instancia.
4. Declarar una clase como static en java se hace de forma anidada. Esto quiere decir que existe una clase exterior que engloba a la clase static.

La clase static de esta forma puede ser instanciada sin necesidad de instanciar la clase exterior. También, tiene la posibilidad de acceder a todos los otros atributos y métodos miembro de la clase exterior (mientras estos también sean static).

Respuesta 2, la elección de esta se debe a que es la que representa el concepto explicado acerca de las clases static.

Point es una clase static que se encuentra anidada en la clase Line, por lo tanto, es un miembro static de Line. Esta característica hace que no sea necesario instanciar Line para usar Point (descarta la respuesta 4). A su vez, Point tampoco existe por si sola, porque es un miembro de Line (descarta la respuesta 1). Esto deja a la respuesta 2 como la correcta.

Interfaces:

1. Si, todos los métodos en una interfaz son por default públicos. Esto tiene la ventaja de ser visibles a cualquier clase que implemente la interfaz.
2. Sería necesario porque como las interfaces definen comportamientos que luego serán comunes a probablemente más de una clase, es posible que los métodos a utilizar necesiten de cierto valor constante, que no quedaría como un buen diseño si hubiera que repetirlo a cada clase.
3. Los métodos por defecto se introdujeron ya que se encontró con la dificultad de adicionar métodos a las interfaces, cosa que requería cambiar también las clases que las implementaban para que también usaran esos métodos nuevos. Con los métodos por defecto las interfaces pueden declarar implementaciones para estos métodos sin que esto afecte a las clases que las utilizan.

Clases abstractas e interfaces:

1. No es posible extender más de una clase en Java.
2. No hay un límite de interfaces que puede implementar una clase.
3. La extensión establece una relación de tipo “es un”, por ejemplo, un Perro es un Animal. La extensión, además, es una relación de herencia por lo que existe una clase Padre (la que es extendida) y una Clase hijo (la que extiende). La implementación en cambio es una relación de tipo “utiliza o implementa”, por ejemplo, un Perro Corre, pero una Persona también Corre. Las interfaces, por lo tanto, definen un comportamiento que puede ser común a muchas clases.

Herencia:

1. La herencia es una relación de tipo jerárquica entre clases, leyéndose de las clases hijas a las padres de forma tal que se dice que una clase hija es una clase padre. Por ejemplo: un Televisor es un Electrodoméstico.
2. Creo que está relacionado con la extensibilidad.
3. Creo que está relacionado con el polimorfismo.

Polimorfismo:

1. Polimorfismo es la habilidad de un objeto para tomar diversas formas, es decir para tomar diferentes comportamientos en diferentes situaciones.
2. Si no existiera el polimorfismo no se podría sobrecargar métodos ni sobrescribirlos, ya que solo tendrían una forma y esto iría en contra de la reutilización del código. Tampoco se podría asignar una clase hija a una referencia de una clase padre.
3. Parametro formal es aquel especificado en el prototipo de un método, y el parámetro actual es el valor que tiene un parámetro formal en un momento dado.