COMPLEMENTOS DE PROGRAMACIÓN

EXAMEN JUNIO 2014

1) **Explica qué problema hay en el código dado:**

El problema es que al crear el objeto a de la clase A en el main no le estamos pasando como parámetro del constructor nada, cuando deberíamos de pasarle un String. Por tanto nos dará un error de compilación.

Podemos añadir también que no es recomendable crear un constructor que tenga como parámetro una variable que se llame igual que el dato miembro creado.

2) **Muestra la salida del programa dado:**

La salida es:

contador vale 101

nVeces vale 0

Esto se debe a que el número que le pasamos al método como nVeces es pasado por valor, es decir que se hace una copia del elemento original y se modifica el valor copiado mientras que el valor original queda intacto.

Sin embargo el elemento c es pasado por referencia ya que le estamos pasando la dirección de memoria donde se encuentra el valor original, por tanto se modifica el valor original.

3) **Indica si el código dado compila bien o no, o si da algún error de ejecución y por qué.**

El código compilaría bien pero da un error de ejecución al hacer la llamada "System.out.println(dates[0].toString())" porque hemos inicializado un vector java.util.Date con tamaño 10 pero no hemos escrito ningún valor en él por lo que no hay ningún valor que escribir con toString().

4) **Indica si los siguientes dos programas compilan, y en caso de que sea así, cuál es su salida al ejecutarlos. Si no compila explicar por qué.**
    a) **Programa 1.**

    Este programa si compila bien y su salida es:

    Se llama al constructor por defecto de la clase A

    b) **Programa 2.**

    No compilaría bien porque al ser la clase B una clase heredada de A, cuando en el main se crea un objeto de la clase B lo primero que se hace es ir a la clase B y seguidamente ir a la clase A para ejecutar el constructor de la clase A. Sin

embargo como al constructor de la clase A se le pasa un parámetro y al crear el objeto con su constructor no se le ha pasado ningún parámetro no nos dejará que compilemos el programa.

5) Identifica los problemas del código dado:

En primer lugar nos encontramos con una no recomendación que consiste en que no es recomendable poner el mismo del dato miembro a los parámetros de algún método o del constructor, así que si le hemos puesto "radio" al dato miembro, podríamos ponerle "r" al parámetro pasado al constructor. Y a la hora de referirnos al dato miembro podríamos llamarlo con "this.radio".

En segundo lugar, en el constructor de Cilindro no sería posible hacer "Circulo(radio)" sino que tendríamos que poner "super(radio)" para que acceda al constructor de Circulo. Al ocurrir este fallo ya no compilaría el programa y en el método getArea() tampoco se podría utilizar el dato "radio".

También volvemos a encontrarnos con que el parámetro "altura" que se le pasa al constructor de Cilindro se llama igual que el dato miembro.

Por último en el método getArea() de Cilindro comprobamos que la operación para obtener el área del cilindro no es la correcta. Además entraría en un bucle infinito al llamar a getArea() de nuevo. Debería ser:

Return 2 * super.getArea() * this.altura;

6) Indica cuál es la salida de los programas dados y explica por qué es así.

a) Programa 1.

La salida del programa es:

Tipo de persona: Persona

Tipo de persona: Estudiante

Se crea una variable p de tipo Persona. En primer lugar se crea un objeto apuntando a la clase Persona, por lo que al hacer "p.printPersona()" se va a ejecutar el método printPersona() de la clase Persona y el método getInfo() de la misma clase. En segundo lugar se crea un objeto que apunta a la clase Estudiante(), luego como Estudiante es una clase heredada de Persona, podrá ejecutar los métodos públicos de dicha clase, luego podrá ejecutar el método printPersona(). En este caso como en la clase Estudiante está sobrescrito el método getInfo() y el objeto apunta a dicha clase se ejecutará este último método haciendo que muestre "Estudiante" en lugar de "Persona".

b) Programa 2.

La salida del programa es:

Tipo de persona: Persona

Tipo de persona: Persona

Esto es porque en este caso los métodos tanto en la clase Persona como en la clase Estudiante son privados, por tanto cuando desde el método printPersona() se llama al método getInfo() siempre se ejecutará el de la clase Persona ya que son dos métodos diferentes pero con el mismo nombre, es decir, no están sobrescritos. Para llamar al de la clase Estudiante deberíamos poner "Estudiante.getInfo()".

7) **Explica qué son las excepciones comprobadas (checked exceptions) y las no comprobadas.**

Las excepciones comprobadas son aquellas que se espera que el programa identifique y arregle en el programa. Se generan por causas externas que pueden alterar el programa en tiempo de ejecución. Este tipo de excepciones hacen que el compilador obligue al programador que las trate con un bloque try-catch o bien que las declare en la cabecera del método.

Las excepciones no comprobadas proceden de errores del código y son difíciles de identificar porque el compilador o la JVM no le está indicando en algún momento al programador que la verifique o que haga algo para corregirla cuando se produce. Son los errores en tiempo de ejecución y los errores de código.

8) **Tanto el método Thread.sleep(tiempo) como el método Object.wait() hacen que una hebra detenga su ejecución, pero ¿cuáles son sus semejanzas y diferencias entre ellos?**

En el primer caso la hebra se duerme un intervalo de tiempo especificado en milisegundos, es decir pasa a estar en estado de espera. Cuando dicho tiempo pasa, la hebra despierta y vuelve al estado de ejecución.

Sin embargo en el segundo caso la hebra pasa a estado de espera hasta que otra hebra le notifica que puede volver al estado de ejecución por medio del método notify() o notifyAll().

9) **Dado el código de Java, escribe una clase que permita guardar una colección de objetos cuya clase pueda ser cualquier subclase Figura. Para guardar la colección, tendrá que usarse un array como dato miembro. Debes hacer al menos lo siguiente:**
   a) **Declaración de los dato miembro necesarios.**
   b) **Un constructor por defecto para crear una colección con capacidad inicial para 100 elementos.**
   c) **Un constructor con un parámetro para crear una colección con la capacidad especificada por el parámetro.**
   d) **Un método para obtener el número de elementos que tenemos en la colección.**
   e) **Un método para recuperar el elemento de la posición i. Se valorará positivamente la solución adoptada para el caso de que la posición no sea correcta (posición < 0 || posición >= size()).**

f) **Un método para añadir un nuevo objeto (de cualquier subclase Figura) en la posición final de la colección. En caso de que la capacidad de la colección ya esté agotada, debe crear un nuevo array con el doble de capacidad.**
g) **Un método para añadir un nuevo objeto (de cualquier subclase Figura) en una posición (int) especificada. En caso de que la capacidad de la colección ya esté agotada, debe crear un nuevo array con el doble de la capacidad. Se valorará positivamente la solución adoptada para el caso de que la posición no sea correcta (posición < 0 || posición >= size ()).**
h) **Un método para imprimir en la salida estándar el área de todos los elementos de la colección. Para ello, añade también a las siguientes clases el código necesario para poder obtener el área.**

10) **Implementa, haciendo uso de los métodos wait() y notifyAll() de la clase Object, el problemas de las dos hebras que acceden concurrentemente a una cuenta bancaria (una hebra ingresa dinero en la cuenta y la otra retira dinero de ella). No esta permitido el uso de objetos Lock ni Condition.**
    a) **Las dos hebras ejecutan las siguientes tareas:**
        i) **TareaDepositar: deposita cantidades (aleatorias entre 1 y 10) de dinero en la cuenta. Tras cada depósito, esta tarea debe esperar un segundo.**
        ii) **TareaRetirar: retira cantidades (alatorias entre 1 y 10) de dinero de la cuenta.**
    b) **La tarea TareaRetirar debe esperar si la cantidad a retirar es mayor al balance actual.**
    c) **Cuando se deposita una nueva cantidad, la TareaDpositar debe pedir a la tarea TareaRetirar que intente continuar.**
    d) **Si la cantidad depositada no es suficiente todavía para el reintegro, la TareaRetirar debe seguir esperando a un nuevo depósito.**

El programa constaría una clase Cuenta (interfaz), una clase CuentaImpl (clase que implementa los métodos de la interfaz) y una clase CooperacionHebras (contiene las tareas y el método principal). He seguido el modelo de Productor/Consumidor con la interfaz.

Clase Cuenta:

```
public interface Cuenta{

   public void depositar(int cantidad) throws InterruptedException;

   public void retirar(int cantidad) throws InterruptedException;

}
```

//

Clase CuentaImpl:

```
public class CuentaImpl implements Cuenta {
   private int balance = 0;

   public int getBalance(){
      return balance;
   }
//  Metodo que sirve para aumentar el balance una cierta cantidad pasada como
//  parametro. Tiene que ir en un bloque try-catch
   @Override
   public synchronized void depositar(int cantidad) throws InterruptedException{
      try{
         balance += cantidad;
         System.out.println("Deposito " + cantidad);
         notifyAll();
      }catch(Exception e){
         System.out.println("Excepcion en depositar");
      }
   }
//  Metodo que sirve para disminuir el balance una cierta cantidad pasada como
//  parametro. Tiene que ir en un bloque try-catch por utilizar wait()
   @Override
   public synchronized void retirar(int cantidad) throws InterruptedException{
      try{
         while(cantidad > balance){
            System.out.println("No puedo retirar dinero");
            wait();
         }
```

```
            balance -= cantidad;
            System.out.println("Retiro " + cantidad + "     " + "Queda " + getBalance());
            notifyAll();
        }catch(InterruptedException e){
            System.out.println("Excepcion en tarea retirar " + e.toString());
        }
    }
}
////////////////////////////////////////////////////////////////////////////////////////////
```

Clase CooperacionHebras:

```
import java.util.concurrent.*;
public class CooperacionHebras {
    private final static Cuenta cuenta = new CuentaImpl();

    public static void main(String[] args){
//      Creamos 2 hebras
        ExecutorService executor = Executors.newFixedThreadPool(2);
//      Ejecutamos las hebras con las tareas seleccionadas
        executor.execute(new TareaDepositar());
        executor.execute(new TareaRetirar());
        executor.shutdown();
    }
    private static class TareaDepositar implements Runnable{
        @Override
        public void run(){
            while(true){
                try {
                    cuenta.depositar((int) (Math.random() * 10) + 1);
```

```
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
                System.out.println("Excepcion en TareaDepositar");
            }
        }
    }
    private static class TareaRetirar implements Runnable{
        @Override
        public void run(){
            while(true){
                try{
                    cuenta.retirar((int)(Math.random() * 10) + 1);
                }catch(InterruptedException e){
                    System.out.println("Excepcion en TareaRetirar");
                }
            }
        }
    }
}
```

11) Responde a las siguientes preguntas (Pregunta opcional para subir nota) (FALTAN COSAS Y NO SE SI ESTAN BIEN).

a) ¿Cuáles son los beneficios de usar tipos genéricos?

Permiten detectar errores en tiempo de compilación en lugar de en tiempo de ejecución. No pueden utilizarse tipos primitivos como tipos genéricos. Una colección que use genéricos, no necesita el uso de castings para recuperar uno de los elementos de la colección. Un tipo genérico puede especificarse como subtipo de otro.

b) ¿Cuál es la sintaxis para declarar un tipo genérico en una clase?

Public class NombreClase<E> {

… }

¿Y para hacerlo en un método genérico?

Public Tipo NombreMetodo (E o) {

…

}

c) ¿Qué es un tipo raso (raw type)?

Un tipo raso se crea cuando Java permite que una clase genérica pueda usarse sin argumentos tipo para que un código antiguo pueda funcionar con código genérico.

¿Por qué no es seguro su uso?

Porque se pueden llegar a provocar errores en tiempo de ejecución al comparar dos objetos de diferente tipo.

¿Por qué se permiten en Java?

d) ¿Qué es Erasure?

La información sobre tipos genéricos será borrada una vez que el compilador comprueba que le tipo genérico se usa de forma segura. Esta aproximación permite que le código genérico sea compatible con el código antiguo que usa tipos rasos. Cuando se compilan clases, interfaces y métodos genéricos, los tipos genéricos se sustituyen por el tipo Object.

¿Por qué implementa Java los genéricos usando erasure?