DECSAI
Departamento de Ciencias de la Computación e I.A.

Complementos de Programación

Convocatoria de Junio. Curso 2013/2014

13 de Junio de 2014

1. **(0.5 ptos)** Explica qué problema hay en el siguiente código:

```
1  class A {
2    String s;
3
4    A(String s) {
5      this.s = s;
6    }
7
8    public void print() {
9      System.out.print(s);
10   }
11 }
12 class Prueba {
13   public static void main(String[] args) {
14     A a = new A();
15     a.print();
16   }
17 }
```

2. **(0.5 ptos)** Muestra la salida del siguiente programa:

```
1  public class Contador {
2    public int contador;
3
4    Contador(int c) {
5      contador = c;
6    }
7    Contador() {
8      contador = 1;
9    }
10 }
11 public class Prueba {
12   public static void main(String[] args) {
13     Contador miContador = new Contador();
14     int nVeces = 0;
15
16     for (int i = 0; i < 100; i++)
17       incrementar(miContador, nVeces);
18
19     System.out.println("contador vale " + miContador.contador);
20     System.out.println("nVeces vale " + nVeces);
21   }
22   public static void incrementar(Contador c, int nVeces) {
23     c.contador++;
24     nVeces++;
25   }
26 }
```

3. **(0.5 ptos)** Indica si el siguiente código compila bien o no, o si da algún error de ejecución y por qué.

```
1  public class Prueba {
2     public static void main(String[] args) {
3        java.util.Date[] dates = new java.util.Date[10];
4        System.out.println(dates[0]);
5        System.out.println(dates[0].toString());
6     }
7  }
```

4. **(0.5 ptos)** Indica si los siguientes dos programas compilan, y en caso de que sea así, cual es su salida al ejecutarlos. Si no compila explicar por qué:

   a) Programa 1

```
1  class A {
2     public A() {
3        System.out.println("Se llama al constructor por defecto de la clase A");
4     }
5  }
6
7  class B extends A {
8  }
9
10 public class C {
11    public static void main(String[] args) {
12       B b = new B();
13    }
14 }
```

   b) Programa 2

```
1  class A {
2     public A(int x) {
3        System.out.println("Se llama al constructor con un int de la clase A");
4     }
5  }
6
7  class B extends A {
8     public B() {
9     }
10 }
11
12 public class C {
13    public static void main(String[] args) {
14       B b = new B();
15    }
16 }
```

5. **(0.5 ptos)** Identifica los problemas que hay en el siguiente código:

```
1  public class Circulo {
2     private double radio;
3     public Circulo(double radio) {
4        radio = radio;
5     }
6     public double getRadio() {
7        return radio;
8     }
9     public double getArea() {
10       return radio * radio * Math.PI;
11    }
12 }
13 class Cilindro extends Circulo {
14    private double altura;
15    Cilindro(double radio, double altura) {
16       Circulo(radio);
17       altura = altura;
18    }
19    public double getArea() {
20       return 2 * (getArea() + Math.PI * radio * altura);
21    }
22 }
```

6. **(0.5 ptos)** Indica cuál es la salida de los siguientes programas y explica por qué es así.

   a) Programa 1

```
1  public class Prueba {
2     public static void main(String[] args) {
3        Persona p;
4        p=new Persona();
5        p.printPersona();
6        p=new Estudiante();
7        p.printPersona();
8     }
9  }
10 class Persona {
11    public String getInfo() {
12       return "Persona";
13    }
14    public void printPersona() {
15       System.out.println("Tipo de persona: "+getInfo());
16    }
17 }
18 class Estudiante extends Persona {
19    public String getInfo() {
20       return "Estudiante";
21    }
22 }
```

   b) Programa 2

```
1  public class Prueba {
2     public static void main(String[] args) {
3        Persona p;
4        p=new Persona();
5        p.printPersona();
6        p=new Estudiante();
7        p.printPersona();
8     }
9  }
10 class Persona {
11    private String getInfo() {
12       return "Persona";
13    }
14    public void printPersona() {
15       System.out.println("Tipo de persona: "+getInfo());
16    }
17 }
18 class Estudiante extends Persona {
19    private String getInfo() {
20       return "Estudiante";
21    }
22 }
```

7. **(0.5 ptos)** Explica qué son las excepciones comprobadas (*checked exceptions*) y las no comprobadas.

8. **(0.5 ptos)** Tanto el método `Thread.sleep(tiempo)` como el método `Object.wait()` hacen que una hebra detenga su ejecución, pero ¿cuales son las semejanzas y diferencias entre ellos?.

9. **(3.5 ptos)** Dado el siguiente código Java, escribe una clase que permita guardar una colección de objetos cuya clase pueda ser cualquier subclase de `Figura`. Para guardar la colección, tendrá que usarse **un array como dato miembro**. Debes hacer al menos lo siguiente:

   a) (0.25 ptos) Declaración de los datos miembro necesarios.

   b) (0.25 ptos) Un constructor por defecto para crear una colección con una capacidad inicial para 100 elementos.

   c) (0.25 ptos) Un constructor con un parámetro entero parar crear una colección con la capacidad especificada por el parámetro.

   d) (0.25 ptos) Un método para obtener el número de elementos que tenemos en la colección.

   e) (0.5 ptos)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 (`posicion < 0 || posicion >= size()`).

   f) (0.5 ptos) Un método para añadir un nuevo objeto (de cualquier subclase de `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) (1 pto) Un método para añadir un nuevo objeto (de cualquier subclase de `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 capacidad. Se valorará positivamente la solución adoptada para el caso de que la posición no sea correcta (`posicion < 0 || posicion >= size()`).

   h) (0.5 ptos) 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.

```
abstract class Figura {
  double dim1, dim2;
  Figura(double a, double b) {
    dim1 = a;
    dim2 = b;
  }
}
class Rectangulo extends Figura {
  Rectangulo(double a, double b) {
    super(a, b);
  }
}
class Triangulo extends Figura {
  Triangulo(double a, double b) {
    super(a, b);
  }
}
```

10. **(2.5 ptos)** Implementa, haciendo uso de los métodos `wait()` y `notifyAll()` de la clase `Object`, el problema 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 está permitido el uso de objetos `Lock` ni `Condition`.

    - Las dos hebras ejecutan las siguientes tareas:
        - `TareaDepositar`: deposita cantidades (aleatorias entre 1 y 10) de dinero en la cuenta. Tras cada deposito, esta tarea debe esperar un segundo.
        - `TareaRetirar`: retira cantidades (aleatorias entre 1 y 10) de dinero de la cuenta.
    - La tarea `TareaRetirar` debe esperar si la **cantidad** a retirar es mayor al **balance** actual.
    - Cuando se deposita una nueva cantidad, la `TareaDepositar` debe pedir a la tarea `TareaRetirar` que intente continuar.
    - Si la cantidad depositada no es suficiente todavía para el reintegro, la tarea `TareaRetirar` debe seguir esperando a un nuevo depósito.

11. **(1 pto)** Responde a las siguientes preguntas (Pregunta opcional para subir nota)

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

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

    c) ¿Qué es un tipo raso (*raw type*)? ¿Por qué no es seguro su uso? ¿Por qué se permiten en Java?

    d) ¿Qué es *Erasure*? ¿Por qué implementa Java los genéricos usando *erasure*?

Duración del examen: 2 horas y media.