EXAMEN PDOO : TEORÍA 2 ~ [2019 - 2020]

TIPO DE EXAMEN : 1

Pregunta 1 - [10 puntos]: Dados los siguientes ficheros

```
00: //FICHERO HIJA.JAVA
01: package paqueteA;
02:
03: class Padre{
04:     protected void protegido(){
05:         System.out.println("Protegido Padre");
06:     }
07:     void metodo(){
08:         System.out.println("Metodo Padre");
09:     }
10:     public void procesa(){
11:         System.out.println("Procesando en el padre...");
12:     }
13:     public void ejecutarTarea(){
14:         procesa();
15:         System.out.println("Fin de la tarea en el padre");
16:     }
17: }
18:
19: public class Hija extends Padre{
20:     void test(Padre p){
21:         p.protegido();
22:         p.metodo();
23:     }
24: }
```

```
00: //FICHERO NIETA.JAVA
01: package subpaquete.paqueteA;
02: import paqueteA.Hija;
03:
04: public class Nieta extends Hija{
05:     void test(Hija p){
06:         p.protegido();
07:         p.metodo();
08:     }
09:     void test2(Nieta p){
10:         p.protegido();
11:         p.metodo();
12:     }
13:     @Override
14:     public void procesa(){
15:         System.out.println("Procesando en el nieto...");
16:     }
17:     public void tareaNieto(){
18:         System.out.println("Tarea en el nieto");
19:     }
20: }
```

```java
//FICHERO PRINCIPAL.JAVA
import java.util.ArrayList;
import paqueteA.Hija;
import subpaquete.paqueteA.Nieta;

interface MyInterface{
    public void ejecutarTarea();
}

public class Principal{
    public static void main(String[] args){
        Nieta n=new Nieta();
        n.ejecutarTarea();
        ((Hija) n).ejecutarTarea();

        Hija h=new Nieta();
        MyInterface interf=h;

        h.tareaNieto();

        ArrayList<Integer> array = (ArrayList<Integer>) (Object) h;
        ArrayList<Hija> array2 = new ArrayList<Nieta>();
    }
}
```

Encontrar los errores asociados a la visibilidad/especificadores de acceso. Indicar:

- fichero
- número de linea
- explicación de la causa

Pregunta 2 - [10 puntos]: Dados los siguientes ficheros

```
00: //FICHERO B.JAVA
01: package paqueteA;
02:
03: class A{
04:     protected void protegido(){
05:         System.out.println("Protegido A");
06:     }
07:     void metodo(){
08:         System.out.println("Metodo A");
09:     }
10:     public void procesa(){
11:         System.out.println("Procesando en A...");
12:     }
13:     public void ejecutarTarea(){
14:         procesa();
15:         System.out.println("Fin de la tarea en A");
16:     }
17: }
18:
19: public class B extends A{
20:     void test(A p){
21:         p.protegido();
22:         p.metodo();
23:     }
24: }
```

```
00: //FICHERO C.JAVA
01: package subpaquete.paqueteA;
02: import paqueteA.B;
03:
04: public class C extends B{
05:     void test2(C p){
06:         p.protegido();
07:         p.metodo();
08:     }
09:     @Override
10:     public void procesa(){
11:         System.out.println("Procesando en C...");
12:     }
13:     public void tareaC(){
14:         System.out.println("Tarea en C");
15:     }
16:     void test(B p){
17:         p.protegido();
18:         p.metodo();
19:     }
20: }
```

```
00: //FICHERO PRINCIPAL.JAVA
01: import java.util.ArrayList;
02: import paqueteA.B;
03: import subpaquete.paqueteA.C;
04:
05: interface MyInterface{
06:     public void ejecutarTarea();
07: }
08:
09: public class Principal{
10:    public static void main(String[] args){
11:         ArrayList<Integer> array = (ArrayList<Integer>) (Object) h;
12:         ArrayList<Hija> array2 = new ArrayList<Nieta>();
13:
14:         C n=new C();
15:         n.ejecutarTarea();
16:         ((B) n).ejecutarTarea();
17:
18:         B h=new C();
19:         MyInterface interf=h;
20:
21:         h.tareaC();
22:     }
23: }
```

Encontrar los errores asociados a compatibilidad entre tipos y cuestiones relativas a operaciones de *casting*. Indicar:

- fichero
- número de linea
- explicación de la causa indicando si el error se produce en tiempo de compilación o ejecución.

Pregunta 3 - [8 puntos]: Dados los siguientes ficheros

```
00: //FICHERO B.JAVA
01: package paqueteA;
02:
03: class A{
04:     protected void protegido(){
05:         System.out.println("Protegido A");
06:     }
07:     void metodo(){
08:         System.out.println("Metodo A");
09:     }
10:     public void procesa(){
11:         System.out.println("Procesando en A...");
12:     }
13:     public void ejecutarTarea(){
14:         procesa();
15:         System.out.println("Fin de la tarea en A");
16:     }
17: }
18:
19: public class B extends A{
20:     void test(A p){
21:         p.protegido();
22:         p.metodo();
23:     }
24: }
```

```
00: //FICHERO C.JAVA
01: package subpaquete.paqueteA;
02: import paqueteA.B;
03:
04: public class C extends B{
05:     void test2(C p){
06:         p.protegido();
07:         p.metodo();
08:     }
09:     @Override
10:     public void procesa(){
11:         System.out.println("Procesando en C...");
12:     }
13:     public void tareaC(){
14:         System.out.println("Tarea en C");
15:     }
16:     void test(B p){
17:         p.protegido();
18:         p.metodo();
19:     }
20: }
```

```
00: //FICHERO PRINCIPAL.JAVA
01: import java.util.ArrayList;
02: import paqueteA.B;
03: import subpaquete.paqueteA.C;
04:
05: interface MyInterface{
06:     public void ejecutarTarea();
07: }
08:
09: public class Principal{
10:    public static void main(String[] args){
11:         ArrayList<Integer> array = (ArrayList<Integer>) (Object) h;
12:         ArrayList<Hija> array2 = new ArrayList<Nieta>();
13:
14:         C n=new C();
15:         n.ejecutarTarea();
16:         ((B) n).ejecutarTarea();
17:
18:         B h=new C();
19:         MyInterface interf=h;
20:
21:         h.tareaC();
22:     }
23: }
```

Indicar la salida que se produce en la consola al ejecutar el programa principal ignorando las líneas con errores:

Pregunta 4 - [7 puntos]: Dados el siguiente código

```
package deepspace;
import java.util.ArrayList;

class SpaceAssetsSet{
    private ArrayList<Object> assets=new ArrayList();

    public void add(Object o) {assets.add(o);}
    public ArrayList<Object> getElements() {return assets;}
}
```

```
public class Examen{
    public static void main(String[] args){
        SpaceAssetsSet sa=new SpaceAssetsSet();
        //Se añaden elementos
        ArrayList<Object> assets=sa.getElements();

        float value=0.0f;
        for(Object a:assets){
            if(a instanceof Weapon){
                value += ((Weapon) a).power()*1.5f;
            }
            else{
                if(a instanceof ShieldBooster){
                    value += ((ShieldBooster) a).getBoost()*1.3f;
                }
                else{
                    //Asumimos que tiene que ser una SpaceStation
                    SpaceStation ss=(SpaceStation) a;
                    for(Weapon w::ss.getWeapons()){
                        value += w.power()*1.5f;
                    }
                    for(ShieldBooster s:ss.getShieldBoosters()){
                        value += s.getBoost()*1.3f;
                    }
                    Hangar h=ss.getHangar();
                    if(h!=null){
                        for(Weapon w::h.getWeapons()){
                            value += w.power()*1.5f;
                        }
                        for(ShieldBooster s:h.getShieldBoosters()){
                            value += s.getBoost()*1.3f;
                        }
                    }
                }
            }
        }
    }
}
```

¿Qué problemas encuentras asociados al diseño software proporcionado? Proponer una estrategia de rediseño indicando cómo quedaría el código una vez aplicada. Se puede utilizar código fuente para contestar la pregunta.

EXAMEN PDOO : TEORÍA 2 ~ [2019 - 2020] (SOLUCIONES)

TIPO DE EXAMEN: 1

A continuación, se muestran las soluciones al examen de `TEORÍA2` *de* `PDOO`:

Pregunta 1 - [10 puntos]

SOLUCIÓN: En el código que se nos proporciona es probable que existan más errores de los expuestos aquí. SIn embargo, como dice en el enunciado, SÓLO VEREMOS LOS ERRORES REFERIDOS A ***VISIBILIDAD Y ESPECIFICADORES DE ACCESO****. Aquí la lista:*

- *FICHERO:* `NIETA.JAVA`
  - *Linea 06* : `p.protegido()` → [ERROR DE COMPILACIÓN]. Este método se llama desde un objeto de la clase **Hija**. Este método es **protegido** en la clase **Padre** y se está intentando llamar fuera del paquete donde se declaró, además de que se le pide la ejecución a una instancia de una superclase.
  - *Linea 11* : `p.metodo()` → [ERROR DE COMPILACIÓN]. Este método se llama desde un objeto de la clase **Nieta**. Dicho método tiene **visibilidad de paquete** y no está en el mismo paquete del que se intenta llamar.

PREGUNTA 2 - [10 puntos]

SOLUCIÓN: En el código que se nos proporciona es probable que existan más errores de los expuestos aquí. SIn embargo, como dice en el enunciado, SÓLO VEREMOS LOS ERRORES REFERIDOS A ***COMPATIBILIDAD ENTRE TIPOS Y CASTING****. Aquí la lista:*

- *FICHERO:* `PRINCIPAL.JAVA`
  - *Linea 11* : `ArrayList<Integer> array = (ArrayList<Integer>) (Object) h;`
    - [ERROR EN TIEMPO DE EJECUCIÓN]
    - *Nota: Obviemos que la variable* `h` *no está declarada con anterioridad y no sabemos qué tipo es y en conclusión es un error de compilación. Es una errata del profesor, por lo que supondremos que sí está ya definida.*
    - Esta sentencia siempre compila porque en primer lugar se realiza un *upcasting* a la clase `Object` y luego se realiza un *downcasting* a la clase `ArrayList<Integer>`. El compilador no nos advierte de ningún problema; sin embargo, cuando se ejecute, en tiempo de ejecución se lanzará una excepción. (Si el objeto al que referencia la variable `h` desde un principio es exactamente un `ArrayList<Integer>` no habría ningún problema).
  - *Linea 12* : `ArrayList<Hija> array2 = new ArrayList<Nieta>();`
    - [ERROR EN TIEMPO DE COMPILACIÓN]
    - *Nota: A mi parecer esto debería de ser otra errata porque este código es independiente del anterior. Sin embargo, supongamos que las clases* `Nieta` *e* `Hija` *sí estan definidas.*

        - Esta sentencia produce un error de compilación puesto que los ArrayList no tienen ninguna relación entre sí, independientemente de la relación entre las clases `Hija` y `Nieta`.
    - *Linea 19* : `MyInterface interf=h`
        - [ERROR EN TIEMPO DE COMPILACIÓN]
        - La variable `h` es del tipo estático `B`. Esta clase no implementa la interfaz `MyInterface` ni de forma directa ni indirecta (ya que su ascendiente tampoco la implementa), y por tanto no se puede realizar dicha asignación.
    - *Linea 22* : `h.tareaC()`
        - [ERROR EN TIEMPO DE COMPILACIÓN]
        - El tipo estático de la variable `h` es `B`. Como la clase `B` no posee este método, se produce error de compilación. Para solucionar este problema, bastaría con hacer *downcast* a `C` y se solucionaría el error.

PREGUNTA 3 - [8 puntos]

SOLUCIÓN: Las lineas que arrojan por pantalla resultados son las líneas 15 y 16:

- *Linea 15* : `n.ejecutarTarea();`

```
"Procesando en C..." "Fin de la tarea en A"
```

- Linea 16: `((B) n).ejecutarTarea();`

```
"Procesando en C..." "Fin de la tarea en A"
```

PREGUNTA 4 - [7 puntos]

SOLUCIÓN: El código proporcionado en esta pregunta posee bastantes problemas de diseño, aquí expondre los más graves incluyendo una modificación del diseño en lenguaje natural:

- ERRORES:
    - `ArrayList<Object> assets=sa.getElements();` : Aquí estamos haciendo una copia de identidad del atributo assets de la clase SpaceAssetsSet(). Todo lo que modifiquemos en el array de assets se hará también en el atributo de dicha clase. Para solucionarlo, deberíamos devolver una copia del vector, cambiando el return en la linea 14 y haciendo una copia del vector con assets.clone().
    - `a instanceof <Class>` : Comprobar explícitamente el tipo al que pertenece un objeto está, generalmente muy desaconsejado, puesto que nos limita mucho cuando el volumen de objetos diferentes es relativamente grande.

- CAMBIO DE DISEÑO:
    - Añadir una interfaz llamada SpaceAsset que tenga un método getValue sin implementación. Al estar en java mejor una interfaz ya que además eliminamos la posibilidad de que surjan situaciones de herencia múltiple.
    - Weapon, ShieldBooster, Hangar y SpaceStation realizarían la interfaz. En el caso de Hangar y SpaceStation recorriendo sus contenedores y usando el método getValue de Weapon y ShieldBooster
    - Lo anterior evita tener que usar el acceso al interior de los objetos Hangar y SpaceStation (getWeapons, getShieldBoosters) ya que su implementación de getValue se encarga de todo.
    - Lo números mágicos pasarían a ser atributos de clase de Weapon y ShieldBooster y se usarían dentro de las implementaciones de getValue
    - La clase SpaceAssetsSet cambiaría su `ArrayList<Object>` por `ArrayList<SpaceAsset>`.
    - La propia clase SpaceAssetsSet también podría realizar la nueva interfaz y así, desde el programa principal solo habría que ejecutar el método getValue de esta clase para conseguir el resultado. En SpaceAssetsSet se recorrería el contenedor y se iría ejecutando getValue de sus elementos