

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(); → se ejecuta perfectamente, pues estás en el mismo paquete
22:         p.metodo(); → se ejecuta perfectamente, estás en el mismo paquete
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(); → Error, una Hija no es una Nieta.
07:         p.metodo(); → Error, tiene visibilidad de paquete y está en otro paquete
08:     }
09:     void test2(Nieta p){
10:         p.protegido(); → Se ejecuta perfectamente, es una Nieta y es visible por Nieta.
11:         p.metodo(); → Error, tiene visibilidad de paquete y está en otro paquete
12:     }
13:     @Override
14:     public void procesa(){
15:         System.out.println("Procesando en el nieto...");
16:     }
17:     public void tareaNieta(){
18:         System.out.println("Tarea en el nieto");
19:     }
20: }
```

```
00: //FICHERO PRINCIPAL.JAVA
01: import java.util.ArrayList;
02: import paqueteA.Hija;
03: import subpaquete.paqueteA.Nieta;
04:
05: interface MyInterface{
06:     public void ejecutarTarea();
07: }
08:
09: public class Principal{
10:     public static void main(String[] args){
11:         Nieta n=new Nieta();
12:         n.ejecutarTarea(); → ejecuta bien, usa el código de Padre
13:         ((Hija) n).ejecutarTarea(); → ejecuta bien, resuelve upcast, innecesario y redundante.
14:
15:         Hija h=new Nieta();
16:         MyInterface interf=h; → Error, una Nieta (Hija) no realiza la interfaz (se compila)
17:
18:         h.tareaNieta(); → Error de compilación, Hija no dispone del método "tareaNieta" ni siquiera Rodo
19:
20:         ArrayList<Integer> array = (ArrayList<Integer>) (Object) h; → Error, eu obtuve, una Nieta
21:         → no es una ArrayList de Integer
22:         ArrayList<Hija> array2 = new ArrayList<Nieta>();
23:     }
24: }
```

↳ No puede hacerse dicha conversión.

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(); → Error, visibilidad de paquete
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(); → Error, no es hijo
18:         p.metodo(); → Error, visibilidad de paquete
19:     }
20: }
```

```
00: //FICHERO PRINCIPAL.JAVA
01: import java.util.ArrayList;
02: import paqueteA.B;
03: import subpaquete.paqueteA.C;
04:
05: interface MyInterface{ Node realiza la interfaz
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; ¿Qué es h? → Se asume que es un ArrayList
12:         ArrayList<Hija> array2 = new ArrayList<Nieta>(); → Se ejecuta perfectamente porque es una lista de hijos
13:
14:         C n=new C();
15:         n.ejecutarTarea(); → Segundo a la periferia con el código de Padre
16:         ((B) n).ejecutarTarea(); → Opción, redundante e innecesaria pero funciona
17:
18:         B h=new C();
19:         MyInterface interf=h; → Error de compilación, B no realiza la interfaz.
20:
21:         h.tareaC(); → Error de compilación, B no dispone del método "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(); → Error, estoy en otro paquete, no es visible por C.
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(); → B no es un C
18:         //p.metodo(); → Otro paquete
19:     }
20: }
```

```
00: //FICHERO PRINCIPAL.JAVA
01: import java.util.ArrayList;
02: import paqueteA.B;
03: import subpaquete.paqueteA.C;
04:
05: interface MyInterface{ → Nada realiza la interfaz.
06:     public void ejecutarTarea();
07: }
08:
09: public class Principal{
10:     public static void main(String[] args){
11:         //ArrayList<Integer> array = (ArrayList<Integer>) (Object) b;
12:         ArrayList<Hija> array2 = new ArrayList<Nieta>();
13:
14:         C n=new C();
15:         n.ejecutarTarea(); → "Procesando en C" "Fin de la tarea en d" → No existe.
16:         ((B) n).ejecutarTarea(); → "Procesando en C" "Fin de la tarea en d"
17:
18:         B h=new C();
19:         //MyInterface interf=h; → No existe h.
20:
21:         //h.tareaC(); → No existe h.
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;} → Poco profundo
}
```

```
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;
                        }
                    }
                }
            }
        }
    }
}
```

} Agruparlo en una clase abstracta

?

¿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.

El problema es

• Usar `instanceof` cuando la herencia clásica ya soluciona el problema

Diagrama jerárquico:

- Weapon
- ShieldBooster
- HangarElement
- SpaceStation

→ No destaca que queda el problema de SpaceStation pues no tiene donde que ver con los otros dos

En el git de Sashas aparece un mejor alleg