Práctica Nº 16

Casting de Tipos Primitivos - Polimorfismo

1. Cree un proyecto llamado Practico16 y codifique en él una clase llamada **`TestCasting`** que permita probar en su método main() los siguientes casos de *casting*.

   Para las siguientes expresiones de tipos primitivos, indique si los "casteos" realizados son necesarios o no, y si son legales o no. En caso de error de compilación, proponga una solución y, para los casos en que el código compile, determine qué valor es almacenado.

a.
```
int c = (int) 10.8;
```
Necesario. Si el casting no se hiciese, no podría realizarse la asignación.

b.
```
double d = (double) 10;
```
Innecesario. El widening es automático.
double d = 10;

c.
```
double d = (int)10;
```
Innecesario. No da error de compilación.
double d = 10;

d.
```
int c = (double) 2;
```
Error de compilación. El casting no es legal dado que un double no puede ser convertido en int.
Correcto:
int c = 2;

e.
```
float pi = 3.14;
```
Error de compilación. 3.14 es um double no um float. Dos opciones:
float pi = 3.14f;
float pi = (float) 3.14f;

f.
```
int num1 = 53;
int num2 = 47;
byte num3;
num3 = (num1 + num2);
```
Error de compilación en la asignación a num3 dado que int no puede ser convertido en byte.
Correcto:
nNum3 = (byte) (num1+num2);

g.
```
int num1 = 53;
int num2 = 47;
long num3;
num3 = (num1 + num2);
```
Correcto. El widening de int a long es automático.

h.
```
int miInt;
long miLong =123987654321L; //99L;
miInt = (int) (miLong);
```
Reemplace el valor de miLong por el comentado ¿Se obtiene el mismo resultado?

Necesario. El valor de miInt es -566397263. Si se descomenta el 99L, el valor es 99.

2. Analice el siguiente código e identifique cuál es la salida de los diferentes fragmentos propuestos.

```
public class A{

    int ivar = 7;

    public void m1(){
        System.out.print("A m1, ");
    }
```

```
public class B extends A{
    public void m1(){
        System.out.print("B m1, ");
    }
}

public class C extends B{
```

```
    public void m2(){
        System.out.print("A m2, ");
    }

    public void m3(){
        System.out.print("A m3, ");
    }

}
```

```
public void m3(){
    System.out.print("C m3, "+(ivar + 6));
}
}
```

```
public class Mixed{

    public static void main(String [] args){
        A a = new A();
        B b = new B();
        C c = new C();
        A a2 = new C();

        //Acá irían los fragmentos de código!
    }
}
```

Fragmentos de Código Candidatos	*Salidas*

`b.m1();`
`c.m2();`
`a.m3();`
	`B m1, A m2, A m3,`

`c.m1();`
`c.m2();`
`c.m3();`
	`B m1, A m2, C m3, 13`

`a.m1();`
`b.m2();`
`c.m3();`
	`A m1, A m2, C m3, 13`

`a2.m1();`
`a2.m2();`
`a2.m3();`
	`B m1, A m2, C m3, 13`

3. ¿Cuáles de los pares A-B de métodos permitirán compilar el programa y obtener la salida indicada? (Nota: El método A debe ser insertado en la clase Monstruo, mientras que el método B en la clase Vampiro.)

   Salida esperada:
   una mordida?
   Respirar fuego
   arrrgh

```
public class TestMonstruo {

public static void main(String (I args) {
    Monstruo [] ma = new Monstruo[3];
    ma[O] = new Vampiro();
    ma[1] = new Dragon():
    ma[2] = new Monstruo():
    for(int x = 0: x < ma.length; x++)
        ma[x].aterrorizar(x);
    }
```

```
public class Monstruo{
    //Acá debiera ir el método A
}

public class Vampiro extends Monstruo{
    //Acá debiera ir el método B
}

public class Dragon extends Monstruo{

    boolean aterrorizar(int nivel){
```

```
}                                                   System.out.println("respirar fuego");
                                                    Return true;
                                                }

                                            }
```

1.	A	`boolean aterrorizar(int d) {`
		`  System.out.println("arrrgh");`
		`  return true;`
		`}`
			Funciona correctamente.

	B	`boolean aterrorizar(int x){`
		`  System.out.println("una mordida?");`
		`  return false;`
		`}`	

2.	A	`boolean aterrorizar(int x) {`
		`  System.out.println("arrrgh");`
		`  return true;`
		`}`
			No compila debido al tipo de retorno de aterrorizar en Vampiro.

	B	`int aterrorizar(int f){`
		`  System.out.println("una mordida?");`
		`  return 1;`
		`}`	

3.	A	`boolean aterrorizar(int d) {`
		`  System.out.println("arrrgh");`
		`  return false;`
		`}`
			Compila pero da una salida distinta a la esperada.
			arrrgh
			Respirar fuego
			Arrrgh
			Es importante recordar que la clase Vampiro no sobreescribió el método aterrorizar de Monstruo, sino que creó un método asustar.

	B	`boolean asustar(int x){`
		`  System.out.println("una mordida?");`
		`  return true;`
		`}`	

4.	A	`boolean aterrorizar(int z) {`
		`  System.out.println("arrrgh");`
		`  return true;`
		`}`
			Compila pero da una salida distinta a la esperada.
			arrrgh
			Respirar fuego
			Arrrgh
			Es importante recordar que la clase Vampiro no sobreescribió el método aterrorizar de Monstruo, en esta opción recibe como parámetro un byte en lugar de un int.

	B	`boolean aterrorizar(byte b){`
		`  System.out.println("una mordida?");`
		`  return true;`
		`}`	

4. Defina las clases (nombre, superclase, atributos y métodos) para implementar una solución orientada a objetos para el siguiente problema e implemente en Java.

Un sistema de administración de música permite organizar nuestra colección musical en base a cierta información relevante que contienen los archivos o pistas de audio. Cada pista de la colección posee los siguientes atributos:

- ID
- Título

- Duración (en segundos)
- Artista o Intérprete
- Título del Álbum
- Año
- Género (rock, pop, melódico, etc.)
- Comentarios

Las pistas de música se pueden agregar y eliminar de la colección en todo momento, así como también cambiar cualquiera de los atributos mencionados.

El sistema permite la creación y administración de listas de reproducción o playlists. Una lista de reproducción tiene un nombre que la describe, y consiste en un subconjunto ordenado de la colección. Esto incluye el caso de que una playlist incluya como uno de sus elementos otra playlist. El orden de los elementos de la playlist se puede modificar manualmente.

Además de la funcionalidad mencionada el Sistema debe proveer los siguientes servicios:

- Duración total: El sistema debe ser capaz de calcular la duración total de una playlist creada por el usuario, en base a la suma de las duraciones de los elementos de la misma.
- Cantidad de elementos: El sistema debe ser capaz de contar la cantidad de pistas almacenadas en la colección completa, o en una playlist específica.
- Impresión por pantalla: Se imprime cada uno de los elementos de la misma, en el orden establecido. Cada pista se escribe con el siguiente formato: ID - Título – Artista/Interprete – Álbum (Género, Año) - Duración

La resolución de este ejercicio es muy similar a la obtenida para el problema de Héroes y Villanos. Se conforma de 3 clases: Coleccion, Playlist y Pista. Coleccion es el padre de una jerarquía de la que heredan Playlist y Pista. La Playlist contiene un conjunto de Coleccion. La funcionalidad a implementar son métodos get, los cuales en la Pista retornan los valores de sus atributos directo, mientras que en la Playlist tienen que recorrer todos sus elementos y hacer alguna operación con ellos (análogo al getFuerza() del problema de Héroes y Villanos).