Ejercicio 1:

Utilizando NetBeans crea un proyecto de Java con el nombre UT2.PD6.

(Ver https://netbeans.org/kb/docs/java/quickstart.html, tema Setting Up the Project).

Crea el Package UT6.PD2Auxiliar.

(Ver video https://www.youtube.com/watch?v=a-BkOK1YzjI, hasta el minuto 3).

(Concepto de Package: Aprenda Java como si estuviera en primero, 1.5.4, página 20).

Dentro del mismo crea:

1. una Interface ICanal, con las siguientes firmas:

```
public String getNombre();
public void setNombre(String nombre);
public <T> T[] getProductos();
```

(Concepto de Interface: Aprenda Java como si estuviera en primero, 1.5.3, página 20).

2. una clase Producto, con la siguiente variable miembro.

```
private String nombre;
```

(Concepto de Clase: Aprenda Java como si estuviera en primero, 1.5.1, página 20).

Implementar los getter, setter de la misma.

Esto puede hacerse de forma automática y tecleando muy pocas líneas de código (ver https://netbeans.org/kb/docs/java/editor-codereference.html#generatecode temas Smart Code Completion y Genereting Code).

La clase debe poseer un constructor, que reciba por parámetro una cadena de caracteres que será utilizada para inicializar la variable miembro.

(Concepto de Clase: Aprenda Java como si estuviera en primero, 1.5.1, página 20)

Conceptos de POO relacionados al Ejercicio 1

Tipos: Un tipo es un conjunto de operaciones que determinan los mensajes que pueden ser enviados a los objetos de ese tipo.
Los mismos pueden ser declarados explícitamente usando interfaces e implícitamente usando clases.

Encapsulación: Se denomina encapsulación al ocultamiento del estado, es decir, de los datos miembros de un objeto de manera que solo puedan cambiar mediante las operaciones definidas para ese objeto.

Ejercicio 2:

Dentro del Package UT6.PD2, creado automáticamente al iniciar el proyecto, implementa:

1. Dos clases: SeguroAutomotriz y SeguroHipotecario. Ambas heredarán de la clase Producto. Esto se realizará de la siguiente manera. En la declaración de la clase y a continuación de la misma incluye la palabra clave *extends*, como se muestra en el siguiente ejemplo.

```
public class SeguroAutomotriz extends Producto {
```

El siguiente video puede brindar una idea básica de cómo funciona la herencia en Java (ver https://www.youtube.com/watch?v=CIxl2vdCi_4).

Al agregar la palabra clave *extends*, NetBeans indicará un error. ¿Puedes indicar que tipo de error es?
Si no es así te recomendamos que vuelvas a leer:

(Depuración: Pensando la computación como un científico, 1.5.3, página 303).

Para solucionar el error recurre al siguiente material de la unidad temática.

(Cómo funcionan los packages: Aprenda Java como si estuviera en primero, 3.6.2, página 45).

Luego de solucionar el error relacionado a los packages, NetBeans indicará un nuevo error. ¿Puedes indicar que tipo de error es?
Para solucionar el error recurre al siguiente material de la unidad temática.

(Constructores en clases derivadas: Aprenda Java como si estuviera en primero, 3.7.5, página 47).

En esta página puedes ver cómo se implementan los constructores de clases heredadas (https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=653:ejemplo-de-herencia-en-java-uso-de-palabras-clave-extends-y-super-constructores-con-herencia-cu00686b&catid=68&Itemid=188).

Este error puede solucionarse sin teclear una línea de código. Pulsa el ícono con forma de lámpara ubicado a la izquierda de la declaración de la clase y selecciona la opción "Add Constructor".

2. Luego de solucionar los errores, implementa en la clase SeguroAutomotriz la variable miembro:

   ```
   private String condicionesDeAsegurabilidad;
   ```

   y en la clase SeguroHipotecario la variable miembro:

   ```
   private String declaracionJurada;
   ```

Implementa los getter y setter de las mismas, tal como lo hiciste en el ejercicio 1.2.

3. Agrega en ambas clases una sobrecarga del constructor, que reciba por parámetro dos cadenas de caracteres. Las mismas serán utilizadas para inicializar las variables miembro (la de la superclase y la agregada en el punto 2 de éste ejercicio).

Esto puede hacerse de forma automática y sin escribir una línea de código (ver https://netbeans.org/kb/docs/java/editor-codereference.html#generatecode tema Genereting Code, apartado Using the Code Generation Dialog Box).

Conceptos de POO relacionados al Ejercicio 2

Generalización: La generalización es una relación taxonómica entre una clase base más general o abstracta y otra clase sucesora más específica o concreta.

Herencia: La herencia es un mecanismo de los lenguajes de programación para implementar declarativamente relaciones de generalización entre una clase base y una o más clases sucesoras.

Sobrecarga: Una operación en un tipo o un método en una clase está sobrecargado cuando hay más de uno don el mismo nombre, pero con diferentes parámetros, o parámetros de diferente tipo.

Ejercicio 3:

Dentro del Package UT6.PD2, implementa:

1. Dos clases: BancoNacion y BancoProduccion. Ambas implementarán ICanal. Esto se realizará de la siguiente manera. En la declaración de la clase y a continuación de la misma incluye la palabra clave *implements*, como se muestra a continuación.

```
public class BancoNacion implements ICanal {
```

Al agregar la palabra clave *implements*, NetBeans indicará un error que ya hemos visto. Soluciónalo de la misma forma que en la parte 1 del ejercicio 2.

2. Luego de solucionado el error, implementa en ambas clases los métodos correspondientes a las firmas de la interface.

Puedes escribir las firmas de la interface o dejar que NetBeans lo haga de forma automática (ver https://blogs.oracle.com/netbeansphp/hint:-implement-all-abstract-methods. El ejemplo presentado corresponde a herencia (utiliza la palabra clave extends, en lugar de implements), pero funciona de la misma forma para la implementación de las firmas de una interface).

NetBeans agrega las firmas de los métodos e incluye un código por defecto en el cuerpo de los mismos. ¿Podrías explicar qué es lo que hace este código?

3. Agrega en ambas clases la siguiente variable miembro.

```
private String nombre();
```

Elimina el código por defecto agregado por NetBeans en los métodos getNombre() y setNombre() y agrega al comportamiento correspondiente para el acceso y la modificación de la variable miembro nombre.

4. Agrega en la clase BancoNacion la variable miembro:

```
private SeguroAutomotriz[] productos;
```

Implementar el método getProductos, agrega un nivel de dificultad. Éste utiliza Generics de Java. Como esta es una primera aproximación a Generics se explicará en detalle que es lo que tendrás que hacer.
Te recomendamos que leas la documentación de Oracle sobre Genrics, ya que lo utilizarás a lo largo del curso: https://docs.oracle.com/javase/tutorial/java/generics/index.html

Cuando declaramos la firma de getProductos en la interface, lo hicimos de la siguiente manera:

```
public <T> T[] getProductos();
```

Al agregar <T>, lo que hicimos fue decirle al programa que la implementación de este método tendrá que lidiar con una clase del tipo T (genérica). Luego, al indicar que retornaría el método, le dijimos que sería un Array de ese tipo T.

Al hacerlo, dejamos abierta la posibilidad de que la implementación del método utilizara cualquier clase que el desarrollador desee.

Para cambiar la firma de acuerdo con nuestros requerimientos, sustituiremos la clase genérica por la que deseamos usar, por lo que la firma de nuestro método quedará:

```
public SeguroAutomotriz[] getProductos();
```

Esto evitará el uso de casting y por ende errores en tiempo de ejecución.

Ahora sólo queda eliminar el código por defecto agregado por NetBeans en el método y sustituirlo por el siguiente:

```
return productos;
```

5. Agrega en la clase BancoProduccion la variable miembro:

   ```
   private SeguroHipotecario[] productos;
   ```

Modifica la firma e implementa el método getProductos, pero ten en cuenta que esta vez productos es un Array del tipo SeguroHipotecario.

6. Agrega en ambas clases un constructor que reciba por parámetros un String y un Array del tipo adecuado, que serán asignados a las variables miembro:

Conceptos de POO relacionados al Ejercicio 3

Composición: La composición es una asociación fuerte entre una clase compuesta y una clase componente en la que instancias de la clase componente no suelen existir independiente de instancias de la clase compuesta.

Ejercicio 4:

1. Dentro de la clase BancoNación agrega el siguiente método:

```
public void imprimirProductos(){
      for (int i = 0; i <= productos.length; i++){
         System.out.println(productos[i].getNombre());
      }
}
```

2. Dentro de la clase BancoProduccion agrega el siguiente método:

```
public void imprimirPrimero(){
      System.out.println(productos[1].getNombre());
}
```

3. Dentro de la clase Main de tu programa, agrega el siguiente código y luego corre el programa:

```
SeguroAutomotriz[] productos = new SeguroAutomotriz[4];
for (int i = 0; i < productos.length; i++){
    productos[i] = new SeguroAutomotriz("Condicion "+i,"Nombre"+i );
}
BancoNacion bancoNacion = new BancoNacion("BancoNacion", productos);
bancoNacion.imprimirProductos();
```

¿Ha sucedido algún error al ejecutar el programa? De ser así, ¿puedes identificar qué tipo de error es?
Te recomendamos que leas nuevamente:

(Depuración: Pensando la computación como un científico, 1.5.3, página 303).

Para encontrar el problema y solucionarlo es conveniente que veas el siguiente video, indicado en los materiales de estudio de la Unidad Temática:
https://www.youtube.com/watch?v=ReLlcbi1es4

4. Dentro de la clase Main de tu programa, agrega el siguiente código y luego corre el programa:

```
SeguroHipotecario[] productos1 = new SeguroHipotecario[4];
for (int i = 1; i < productos1.length; i++){
    productos1[i-1] = new SeguroHipotecario("Condicion "+i,"Nombre "+i );
}

BancoProduccion bancoProduccion = new BancoProduccion("BancoProdicción",
productos1);
bancoProduccion.imprimirPrimero();
```

¿Ha sucedido algún error al ejecutar el programa? De ser así, ¿puedes identificar qué tipo de error es?
¿Retorna este método el primer elemento del Array?