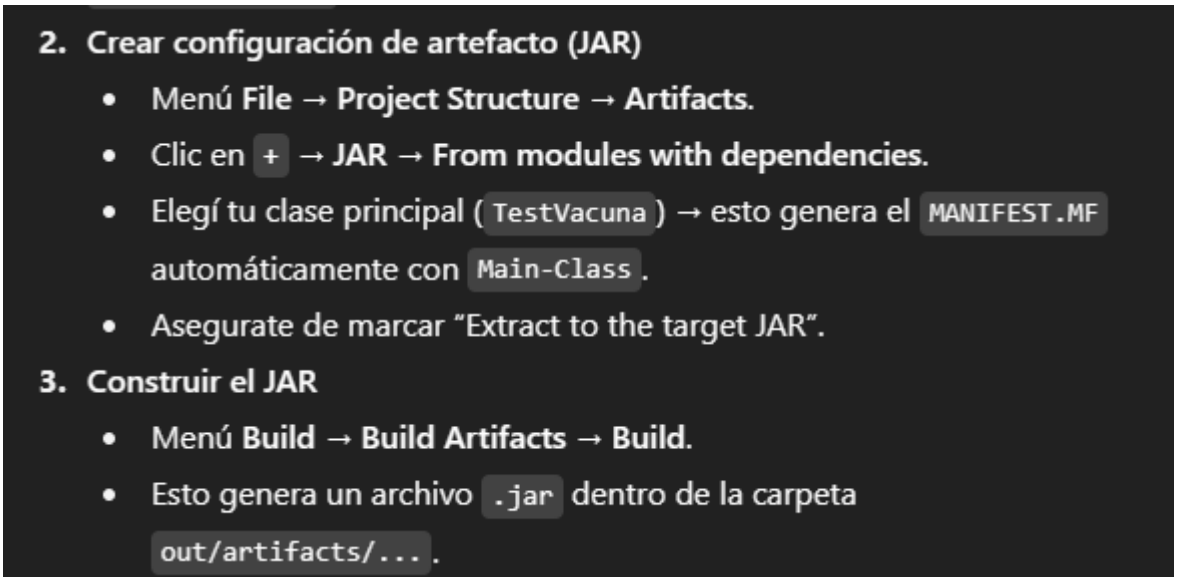
1.
b) Escriba el método main() en la clase TestVacuna, donde se debe crear un arreglo con 5 objetos Vacuna inicializados, para luego recorrer el arreglo e imprimir en pantalla los objetos guardados en él.
c) Comente el método toString() escrito en la clase Vacuna y vuelva a ejecutar el programa. ¿Cuál es la diferencia entre b) y c)?
c. La diferencia es que el default imprime el número del objeto en memoria en hexadecimal. es una representación del objeto.

d) Cree otro objeto de tipo Vacuna y compárelo con el anterior. ¿Qué método de Object es utilizado para la comparación por contenido?.
d. para comparar por contenido habría que sobreescribir equals, sino hace lo mismo que ==.

e) Ejecute la aplicación fuera del entorno de desarrollo. ¿Para que se utiliza la variable de entorno CLASSPATH?
e. con classpath, la jvm sabe de dónde sacar las clases compiladas que se usan.

f) Construya un archivo jar con las clases anteriores, ejecútelo desde la línea de comandos. ¿Dónde se especifica en el archivo jar la clase que contiene el método main?
f.La clase que tiene el método main se especifica en el archivo META-INF/MANIFEST.MF, dentro del JAR, mediante la propiedad Main-Class.
javac Vacuna.java TestVacuna.java
echo "Main-Class: TestVacuna" > manifest.txt
jar cfm Vacunas.jar manifest.txt *.class
java -jar Vacunas.jar
desde la consola, sino



2.- Analice las siguientes clases y responda cada uno de los incisos que figuran a continuación.
a) Considere la siguiente clase Alpha. ¿Es válido el acceso de la clase Gamma?. Justifique.

```
package  griego;
class Alpha {
     protected int x;
     protected void otroMetodoA(){
         System.out.println("Un método protegido");
     }
}

package  griego;
class Gamma {
   void unMétodoG(){
         Alpha a = new Alpha();
         a.x=10;
         a.otroMetodoA();
     }
 }
```

Si, es válido porque están en el mismo paquete. Con protected, permitis que cuaolquier clase dentro del paquete acceda y además pueden acceder subclases de otros paquetes.

b) Considere la siguiente modificación de la clase Alpha. ¿Son válidos los accesos en la clase Beta?. Justifique.

```
package griego;

public class Alpha {
    public int x;
    public void unMetodoA(){
         System.out.println("Un Método Público");
    }
}


package romano;
import griego.*;

class Beta {
   void unMetodoB(){
         Alpha a=new Alpha();
         a.x=10;
         a.unMetodoA();
   }
}
```

Si, el acceso es válido porque public permite que se acceda desde cualquier paquete.

c) Modifique la clase Alpha como se indica debajo. ¿Es válido el método de la clase Beta?. Justifique.

```
package  griego;
public class Alpha {
    int x;
    void unMetodoA(){
         System.out.println("Un mét. paquete");
    }
}

package romano;
import griego.*;
class Beta {
   void unMetodoB(){
         Alpha a = new Alpha();
         a.x=10;
         a.unMetodoA();
    }
}
```

No es válido porque si no se le pone nada es por defecto package y la clase beta está en otro paquete

d) Considere el inciso c) ¿Es válido el acceso a la variable de instancia x y al método de instancia unMetodoA() desde una subclase de Alpha perteneciente al paquete romano?. Justifique.

No, no es válido porque no está en el mismo paquete. tendrían que ser public o protected. Al heredar solo se puede acceder a las cosas public o protected de la clase, el resto no son heredables.

e) Analice el método de la clase Delta. ¿Es válido? Justifique analizando cómo influye el control de acceso protected en la herencia de clases.

```
package  griego;
public class Alpha {
     protected int x;
     protected void otroMetodoA(){
         System.out.println("Un método protegido");
     }
}

package romano;
import griego.*;
public class Delta extends Alpha {
   void unMetodoD(Alpha a, Delta d){
          a.x=10;
          d.x=10;
          a.otroMetodoA();
          d.otroMetodoA();
         }
}
```

No, ya que puede acceder a las cosas de d pero no a las de a. cuando usamos protected, las subclases que estén en otros paquetes tienen acceso, pero no se tiene acceso a la clase de por si.

3.1

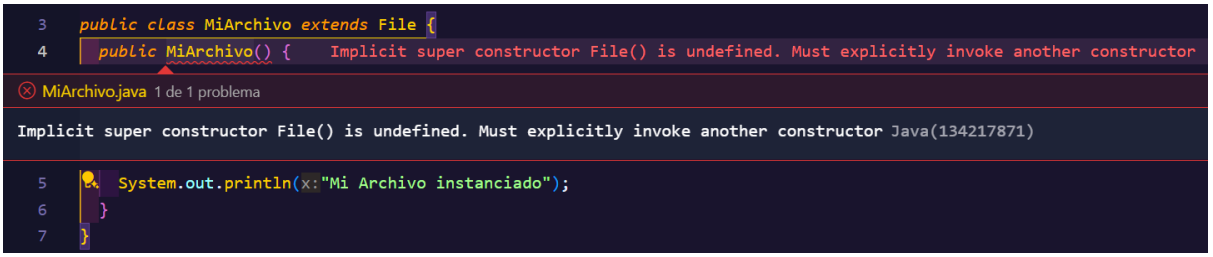
c) ¿Pudo compilar las clases? ¿Qué problemas surgieron y por qué? ¿Cómo los solucionó?

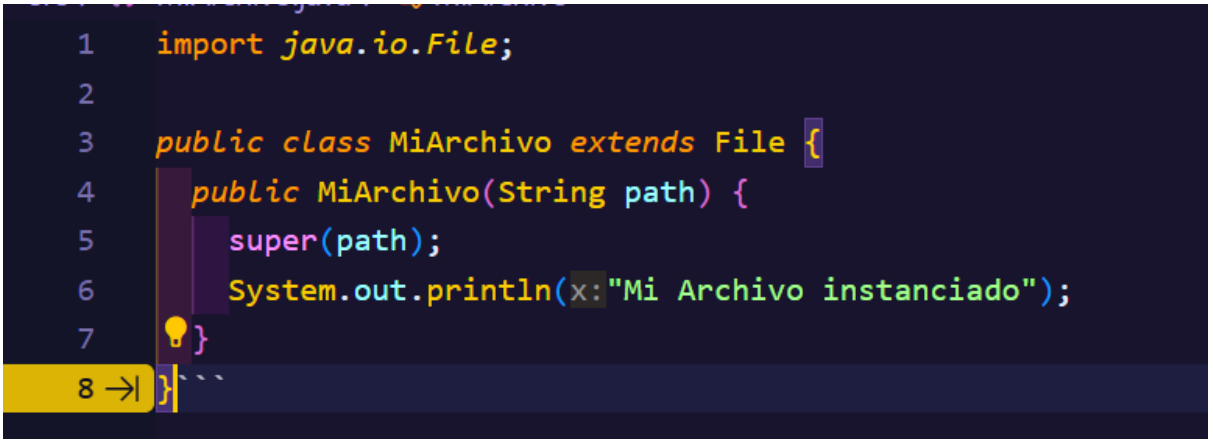
c-Al implementar las subclases, surge el problema de compilación si la clase Vacuna no define un constructor por defecto. Esto ocurre porque en Java, cuando se crea un objeto de una subclase, primero se debe ejecutar un constructor de la superclase.
La solución es proveer en Vacuna un constructor vacío o bien llamar explícitamente a super(...) con los parámetros adecuados en los constructores de las subclases.

3.2.- El siguiente código, define una subclase de java.io.File. Verifique si compila. Si no lo hace implemente una solución.

```
package laboratorio;
import java.io.File;
public class MiArchivo extends File {
```

```
    public MiArchivo() {
        System.out.println("Mi Archivo instanciado") ;
    }
}
```





la clase java.io.File no tiene un constructor por defecto. La solución es definir en la subclase un constructor que invoque explícitamente a uno de los constructores disponibles en File, por ejemplo pasando como argumento la ruta del archivo.

3.3.- Las clases definidas a continuación establecen una relación de herencia. La implementación dada, ¿es correcta?.

Constructores privados

```
package laboratorio;
public class SuperClase {
  private SuperClase() {
  }
}

package laboratorio;
public class SubClase extends SuperClase {
      public SubClase() {
  }
}
```

Constructores protegidos

```
package laboratorio;
public class SuperClase{

 protected SuperClase(){
 }

}

package laboratorio1;

public class SubClase extends SuperClase {
  public SubClase() {
  }
}


package laboratorio1;

public class OtraClase {
  public OtraClase() {
  }
  public void getX() {
      new SuperClase();
  }
}
```

Constructores privados: No es posible heredar de una clase que solo tiene constructores privados, porque la subclase no puede invocar al constructor de la superclase.

Constructores protegidos: Una subclase puede invocar al constructor protegido de la superclase aunque esté en otro paquete, pero otras clases que no son subclases no pueden acceder a ese constructor. Por eso `SubClase` funciona y `OtraClase` no.