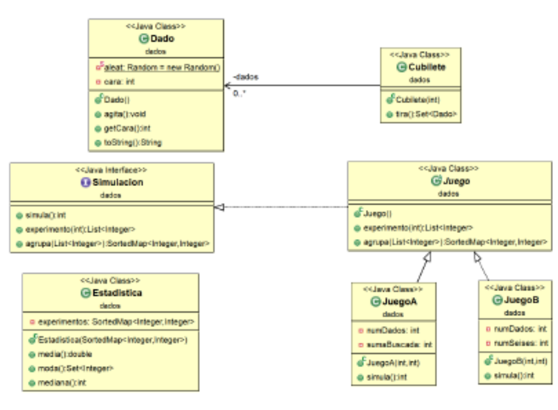
NOTAS PARA LA REALIZACIÓN DEL EJERCICIO:

- Al **inicio del contenido de cada fichero** realizado deberá aparecer un comentario con tus **apellidos y nombre, titulación y grupo.**
- Los diferentes apartados tienen una determinada puntuación. Si un apartado no se sabe hacer, *no debes pararte en él indefinidamente*. Puedes abordar otros.
- **Está permitido:**
  - Consultar los apuntes (CV), la API (Internet), la guía rápida de la API (CV).
  - Añadir métodos privados a las clases.
- **No está permitido**:
  - Intercambiar documentación con otros compañeros.
  - Recibir ayuda de otras personas. Se debe realizar personal e individualmente la solución del ejercicio propuesto.
  - Añadir métodos no privados a las clases.
  - Añadir variables o constantes a las clases.
  - Modificar la visibilidad de las variables, constantes y métodos que aparecen en el diagrama UML.
  - Modificar el código suministrado.
- Una vez terminado el ejercicio, debéis subir (a la tarea creada en el campus virtual para ello) un fichero comprimido de la carpeta **src** que hayáis realizado y usáis vuestros apellidos y nombre para el nombre del mismo **(Apellido1Apellido2Nombre.rar o .zip)**.
- La evaluación tendrá en cuenta la claridad de los algoritmos, del código y la correcta elección de las estructuras de datos, así como los criterios de diseño que favorezcan la reutilización.
- Para la corrección del ejercicio se utilizarán **programas de detección de copias/plagios**.
- Con posterioridad a la realización del ejercicio, el profesor podrá convocar a determinado/as alumno/as para realizar **entrevistas personales síncronas** con objeto de comprobar la autoría de las soluciones entregadas.

Proyecto prDados

Se va a crear una aplicación para simular diferentes juegos con dados. Para ello, se crearán las clases `Dado`, `Cubilete`, `Juego`, `JuegoA`, `JuegoB`, `Estadistica`, `MainCubilete`, `Main` y la interfaz `Simulacion`.

- La interfaz y las clases se crearán en el paquete `dados`, excepto aquellas cuyo nombre comienzan por `Main`, que se crearán en el paquete "por defecto".
- A no ser que se indique lo contrario, las variables de instancia serán privadas y los métodos públicos.



WUOLAH

Clase `Dado` (0.5 ptos)

Crea la clase `Dado` que mantiene información de la cara que muestra un dado cuando está posado. Como **variable de instancia** tendrá un entero que indica el valor de la cara que muestra el dado. (0.1) El constructor debe crear el dado que almacene el valor (generado aleatoriamente del 1 al 6, ambos inclusive) que representa la cara superior del dado. Para ello, la clase dispondrá de una **variable de clase** que referencie a un objeto generador de números aleatorios.

- (0.2) El método `void agita()` simula que se tira el dado, por lo que almacenará en la variable de instancia un nuevo valor generado aleatoriamente entre el 1 y el 6, ambos inclusive, que representa el valor de la cara superior del dado.
- (0.1) El método `int getCara()` devuelve el valor almacenado en la variable de instancia que representa el valor de la cara superior del dado.
- (0.1) El método `String toString()` devuelve la representación textual del objeto actual, con el valor de la cara entre corchetes. Por ejemplo, "`[3]`" (sin comillas).

Clase `Cubilete` (1 pto)

Crea la clase `Cubilete` que contiene varios dados. Los dados se guardarán en una lista.

- (0.5) El constructor recibirá como parámetro el número de dados que contendrá el cubilete, los creará y los almacenará en la lista. Si el valor recibido como parámetro es cero o negativo, entonces lanzará una excepción del tipo `IllegalArgumentException`.
- (0.5) El método `Set<Dado> tira()` simula el agitado y volcado del cubilete. Se comporta agitando (invocando al método agita) todos los dados del cubilete y después devuelve un conjunto con los dados después del agitado. (Los dados no se quitan del cubilete).

NOTA: Debido a que la clase `Dados` no redefine el método `equals(Object obj)`, dos dados son iguales si y solo si referencian al mismo objeto. Por eso, un conjunto puede contener varios objetos dados aunque estos muestren la misma cara.

Clase `MainCubilete` (0.5 ptos)

Crea la aplicación `MainCubilete` que crea un cubilete con 5 dados, lo tira y muestra por consola los valores que han salido en cada dado. Si la cara de algún dado es un 6 se debe mostrar "`Ha salido al menos un 6`". En otro caso se debe mostrar "`No ha salido ningún 6`".

Interfaz `Simulación` (0.25 ptos)

Crea la **interfaz** `Simulacion` que la implementará aquella clase que quiera hacer una simulación de un juego con un cubilete. Esta interfaz exige a las clases que la implementen la definición de tres métodos:

- El método `int simula()` hace una simulación del juego. El resultado de la simulación siempre será un número entero.
- El método `List<Integer> experimento(int numSim)` que realiza `numSim` simulaciones del juego y devuelve una lista de enteros.
- El método `SortedMap<Integer, Integer> agrupa(List<Integer> list)` tomará una lista de enteros con resultados de experimentos y los clasificará agrupándolos mediante algún criterio.

Clase `Juego` (2 ptos)

Crea la **clase abstracta** `Juego` que implementa la interfaz `Simulacion`. La case dejará sin implementar el método `simula` (clase abstracta), y proporcionará una implementación del resto de métodos del modo siguiente:

- (1.00) El método `List<Integer> experimento(int numSim)` que realiza `numSim` simulaciones del juego (invocando al método simula) almacenando en una lista sus resultados. El método devuelve dicha lista. Un argumento menor o igual que 0 en el método `experimento` provocará el lanzamiento de la excepción `IllegalArgumentException.`
- (1.00) El método `SortedMap<Integer, Integer> agrupa(List<Integer> list)` tomará una lista de enteros como la que devuelve el método anterior y contará cuántas veces aparece cada resultado, devolviendo la información obtenida en una correspondencia ordenada. Por ejemplo, si la lista es `[4, 5, 3, 4, 2, 4, 5]` (la primera simulación devolvió el número 4, la segunda devolvió el número 5, …) la correspondencia devuelta será `{2=1, 3=1, 4=3, 5=2}` (el número 2 fue devuelto una vez, el número 3 fue devuelto una vez, el número 4 fue devuelto tres veces y el número 5 fue devuelto dos veces).

Clase `JuegoA` (1 pto)

Crea la clase `JuegoA` como una subclase de la clase `Juego` para simular un juego en el que se tira el cubilete hasta conseguir una cierta suma del valor de los dados. Por ejemplo, suponiendo un objeto de la clase `JuegoA` con un cubilete con 2 dados y suma 7, en la siguiente simulación se necesitan 3 tiradas de los dados hasta conseguir que la suma del valor de los dados sea 7:

- Tirada 1: 3 y 5 (suma igual a 8)
- Tirada 2: 1 y 4 (suma igual a 5)
- Tirada 3: 4 y 3 (suma igual a 7)
- valor devuelto por la simulación: 3

- (0.25) La clase dispondrá de un constructor `JuegoA(int n, int suma)` que recibe como parámetros un número entero `n` que indica cuántos dados se van a introducir en el cubilete que será creado en el método simula y un número entero `suma` que indica la suma deseada que debemos conseguir al tirar los dados del cubilete. El parámetro `n` debe ser un valor mayor que cero, y `suma` un valor entre `n` y `6*n`. Cualquier valor no válido de los argumentos provocará el lanzamiento de una excepción `IllegalArgumentException`.
- (0.75) La implementación del método `simula` que realiza el `JuegoA` es la siguiente:
  - Se crea el cubilete con el número de dados especificado como primer parámetro del constructor.
  - Se tira el cubilete y mientras la suma de los valores que muestran los dados no sea la suma deseada (especificada como segundo parámetro del constructor) se repite el proceso, contando cuántas veces se repite.
  - El valor devuelto resultado de la simulación será el número de veces que ha sido necesario tirar el cubilete hasta obtener la suma deseada.

Clase `JuegoB` (1 pto.)

Crea la clase `JuegoB`, como una subclase de la clase `Juego`, para representar un juego donde la simulación trata de conseguir sacar un mínimo número de seises en las sucesivas jugadas. Por ejemplo, suponiendo un objeto de la clase `JuegoB` con un cubilete con 3 dados y tratando de conseguir 4 seises, en la siguiente simulación se necesitan 4 jugadas hasta conseguir los 4 seises:

- Tirada 1: 3, 6 y 5 (total acumulado de dados iguales a seis: 1)
- Tirada 2: 6, 4 y 6 (total acumulado de dados iguales a seis: 3 = 1+2)
- Tirada 3: 1, 4 y 3 (total acumulado de dados iguales a seis: 3 = 3+0)
- Tirada 4: 2, 6 y 6 (total acumulado de dados iguales a seis: 5 = 3+2)
- valor devuelto por la simulación: 4

En este otro ejemplo con 3 dados y tratando de conseguir 4 seises se necesitan 3 jugadas para conseguir los 4 seises.

- Tirada 1: 3, 6 y 5 (total acumulado de dados iguales a seis: 1)
- Tirada 2: 6, 4 y 6 (total acumulado de dados iguales a seis: 3=1+2)
- Tirada 3: 1, 6 y 3 (total acumulado de dados iguales a seis: 4=3+1)
- valor devuelto por la simulación: 3

- (0.25) La clase dispondrá de un constructor `JuegoB(int n, int numSeises)` que recibe como parámetros un número entero `n` que indica cuántos dados se van a introducir en el cubilete que será creado en el método `simula` y un número entero `numSeises` que indica el mínimo número de seises que se debe conseguir. Argumentos con valor menor o igual que cero provocarán el lanzamiento de una excepción `IllegalArgumentException`.
- (0.75) La implementación del método `simul a` que realiza el `JuegoB` es la siguiente:
  - Se crea el cubilete con el número de dados especificado como primer parámetro del constructor.
  - Se tira el cubilete, y seguirá tirando el cubilete mientras no se haya acumulado una cuenta total (del número de dados cuyos valores sean iguales a seis) igual o superior al valor especificado como segundo parámetro del constructor.
  - El valor devuelto resultado de la simulación será el número de veces que ha sido necesario tirar el cubilete para lograr el total requerido de dados iguales a seis.

Clase `MainPruebaJuegos` (0.5 ptos.)

Crea una aplicación `MainPruebaJuegos` que cree un objeto de `JuegoA` con 2 dados y valor de la suma 7 y llame al método `simula`. Coloca un `System.out.println` en el método `simula` para mostrar el conjunto de dados resultado de cada jugada. Seguidamente haz lo mismo para un objeto de `JuegoB` con 3 dados y 4 seises. Después de probar que tu método `simula` funciona correctamente, **no olvides** de poner en comentario el `System.out.println` anterior.

Clase `Estadistica` (2.5 ptos)

Crea la clase `Estadistica` que mantenga como variable de instancia una correspondencia ordenada como la que devuelve el método `agrupa` de la interfaz `Simulacion` que contiene la información de un experimento. Recordemos que `{2=1, 3=1, 4=3, 5=2}` se interpreta como: en dos jugadas terminó una simulación; en tres jugadas terminó una simulación; en 4 jugadas terminaron 3 simulaciones…).

- (0.10) El constructor recibe como parámetro la correspondencia que debe almacenar.
- (0.70) El método `double media()` calcula y devuelve la media de jugadas de este experimento. En el ejemplo anterior sería (2*1+3*1+4*3+5*2)/(1+1+3+2) = 3.85714286.
- (0.80) El método `Set<Integer> moda()` que devuelve un conjunto con aquellas jugadas que más se repitieron. En el ejemplo anterior será un conjunto con el 4 (se repitió 3 veces). Si la correspondencia fuera `{2=1, 3=1, 4=3, 5=3}`, devolvería un conjunto con los valores 4 y 5 (se repitieron 3 veces). Si la correspondencia fuera `{2=1, 3=1, 4=1, 5=1}` el resultado sería un conjunto con los valores 2, 3, 4 y 5 (se repitieron una vez).
- (0.90) El método `int mediana()` devuelve aquella jugada que queda en el centro cuando se colocan los elementos uno detrás de otro ordenados. En el ejemplo de la correspondencia `{2=1, 3=1, 4=3, 5=2}`, los elementos uno tras otro ordenados serían 2,3,4,4,4,5,5 y por tanto la mediana será el valor central 4. Si el número de elementos fuera par, sería el elemento que ocupa la posición `longitud/2 - 1`.

Clase `MainPruebaEstadistica`

Prueba tu clase `Estadistica` con el siguiente programa:

```
public class MainPruebaEstadistica {
   public static void prueba(SortedMap<Integer, Integer> prueba) {
      Estadistica est = new Estadistica(prueba);
      System.out.println(prueba);
      System.out.println("Media = " + est.media());
      System.out.println("Moda = " + est.moda());
      System.out.println("Mediana = "+ est.mediana());
   }
   public static void main(String[] args) {
      SortedMap<Integer, Integer> map = new TreeMap<>();
      map.put(2, 1);
      map.put(3, 1);
      map.put(4, 3);
      map.put(5, 2);
      prueba(map);
      map.put(5, 3);
      prueba(map);
      map.put(4, 1);
      map.put(5, 1);
      prueba(map);
   }
}
```

Cuyo resultado debe ser

```
{2=1, 3=1, 4=3, 5=2}
Media = 3.857142857142857
Moda = [4]
Mediana = 4
{2=1, 3=1, 4=3, 5=3}
Media = 4.0
Moda = [4, 5]
Mediana = 4
{2=1, 3=1, 4=1, 5=1}
Media = 3.5
Moda = [2, 3, 4, 5]
Mediana = 3
```

Clase `Main` (0.75 ptos)

Crea una aplicación que cree un juego A con un cubilete con un dado y con suma deseada 6. A continuación se realizará un experimento con 30 simulaciones y se mostrará por consola la correspondencia resultado de invocar al método `agrupa` con la lista resultado del experimento. Con la correspondencia resultante se creará una estadística y se mostrarán por consola su media, su moda y su mediana. Por último, se repetirá el proceso, pero con un juego B con un cubilete con 3 dados buscando obtener al menos dos seises.