Practica 2. Ejercicios Juegos

1. Realizar una lectura por pantalla reutilizable de letras y números y un menú de selección de juego que mientras no introduzcamos un número válido nos lo pregunte de nuevo.

UtilidadesJuego.java: leerCadena(), leerNumero().

Aplicacion.java: main(), eligeJuego(), numeroJuegoValido(), preguntaSeguirJugando().

1. Crear la interfaz Jugable y la clase Juego que implementa la interfaz.

Jugable.java: jugar(), muestraNombre(), muestraInfo().

Juego.java: numeroVidasIniciales, numeroVidasRestantes, record, actualizarRecord(), reiniciarPartida(), quitarVida().

1. Crear juego JuegoAdivinarNumero.java que extienda de Juego.java y llamarlo desde el menú de la aplicación

JuegoAdivinarNumero: numeroAdivinar, jugar(), muestraPista(), numeroValido(), muestraNombre(), muestraInfo().

Aplicacion.java: MODIFICAR eligeJuego().

1. Crear juegos JuegoAdivinarPar.java y JuegoAdivinarImpar.java reutilizando el código de JuegoAdivinarNumero.java.

JuegoAdivinarPar.java:

JuegoAdivinarImpar.java:

UtilidadesJuego: esPar(), esImpar(), numeroParAleatorio(), numeroImparAleatorio().

Aplicacion.java: MODIFICAR eligeJuego().

1. Crear JuegoAhorcado.java que extiende de Juego.java

JuegoAhorcado.java: palabraAdivinar, palabraAdivinarOculta, letrasIntroducidas, iniciarJuego(), introducirLetra(), comprobarLetraIntroducidaEnPalabraAdivinar(), comprobarResultadoJuego(), letraValida(), mostrarLetrasIntroducidas(), pintarAhorcado().

Aplicacion.java: MODIFICAR eligeJuego().

Practica 2. Explicación

**UtilidadesJuego.java**

* leerCadena(): Lee una cadena de caracteres desde la entrada estándar del usuario (consola) utilizando un BufferedReader y lo devuelve como una cadena de texto (String). Si la entrada del usuario no es una cadena válida, devuelve “”.
* leerNumero(): Lee un número entero desde la entrada estándar del usuario (consola) utilizando leerCadena() para leer una cadena de texto y luego lo convierte a un entero utilizando Integer.parseInt(). Si la entrada del usuario no es un número válido, devuelve 0.
* esPar(int numero): Verifica si un número entero dado es par, devolviendo true si lo es y false en caso contrario. Utiliza el operador de módulo (%) para determinar si el número es divisible por 2.
* esImpar(int numero): Verifica si un número entero dado es impar, devolviendo true si lo es y false en caso contrario. Utiliza el método esPar(int numero) para determinar si el número no es par.
* numeroParAleatorio(): Genera un número entero aleatorio entre 1 y 10 (inclusive) que sea par. Utiliza el método ints() de la clase Random para generar una secuencia infinita de números enteros, luego filtra aquellos que son pares utilizando el método esPar(int numero), y finalmente toma el primer número par que encuentra utilizando findFirst(). Si no encuentra ningún número par, devuelve 0.
* numeroImparAleatorio(): Genera un número entero aleatorio entre 1 y 10 (inclusive) que sea impar. Utiliza el método ints() de la clase Random para generar una secuencia infinita de números enteros, luego filtra aquellos que son impares utilizando el método esImpar(int numero), y finalmente toma el primer número impar que encuentra utilizando findFirst(). Si no encuentra ningún número impar, devuelve 0.

**Aplicacion.java**

* La clase principal de la aplicación es Aplicacion, que contiene el método main como punto de entrada del programa.
* El programa utiliza un bucle do-while para permitir al usuario jugar varios juegos consecutivamente hasta que decida no jugar más. El bucle se ejecuta mientras el resultado del método preguntaSeguirJugando() sea true.
* Dentro del bucle, se llama al método eligeJuego() para que el usuario elija un juego de una lista de juegos disponibles.
* El método eligeJuego() muestra un menú de opciones de juegos al usuario y lee un número introducido por el usuario utilizando el método leerNumero() de la clase UtilidadesJuego. Luego verifica si el número introducido es válido utilizando el método numeroJuegoValido(), que verifica si el número está entre 1 y 4. Si el número es válido, devuelve el juego correspondiente de la lista juegosDisponibles y sale del bucle.
* El método numeroJuegoValido() verifica si un número dado es mayor que 0 y menor o igual que 4, lo que significa que representa una opción válida de juego.
* Después de salir del bucle, se llama al método preguntaSeguirJugando() para preguntar al usuario si desea jugar otra vez. Este método muestra un mensaje al usuario y lee una cadena de texto utilizando el método leerCadena() de la clase UtilidadesJuego. Luego verifica si la respuesta del usuario es "si" o contiene la letra "s" en minúscula, y devuelve true si es así.
* Finalmente, si el usuario decide no jugar más, se muestra el mensaje "Sesion finalizada" antes de que el programa termine.

**Jugable.java**

* jugar(): Este método no tiene parámetros y no devuelve ningún valor. Representa la acción de jugar en un juego y debe ser implementado por las clases que implementen la interfaz Jugable según su lógica de juego específica.
* muestraNombre(): Este método no tiene parámetros y no devuelve ningún valor. Representa la acción de mostrar el nombre del juego y debe ser implementado por las clases que implementen la interfaz Jugable según la forma en que quieran mostrar el nombre del juego en particular.
* muestraInfo(): Este método no tiene parámetros y no devuelve ningún valor. Representa la acción de mostrar información sobre el juego y debe ser implementado por las clases que implementen la interfaz Jugable según la información que quieran mostrar sobre el juego en particular.

**Juego.java**

* La clase Juego implementa la interfaz Jugable.
* La clase Juego es abstracta, lo que significa que no se puede instanciar directamente, sino que debe ser extendida por clases hijas para ser utilizada.
* La clase Juego tiene tres variables de instancia: numeroVidasIniciales, numeroVidasRestantes y record. numeroVidasIniciales es una constante que se establece en el constructor y representa el número inicial de vidas del juego. numeroVidasRestantes y record son variables que almacenan el número actual de vidas restantes y el récord actual del juego, respectivamente.
* El constructor de la clase Juego toma un parámetro numeroVidasIniciales y lo utiliza para inicializar las variables de instancia numeroVidasIniciales, numeroVidasRestantes y record.
* La clase Juego tiene varios métodos. Entre ellos se encuentran:
* actualizaRecord(): Este método compara el número de vidas restantes con el récord actual y muestra un mensaje si se ha alcanzado o superado el récord.
* reiniciaPartida(): Este método reinicia la partida estableciendo el número de vidas restantes y el récord a sus valores iniciales.
* quitaVida(): Este método resta una vida al número de vidas restantes y muestra un mensaje con el número actual de vidas restantes. Devuelve false si el número de vidas restantes es menor o igual a cero, lo que significa que el jugador ha perdido, y true en caso contrario.
* Varios métodos getter y setter para acceder y modificar las variables de instancia numeroVidasRestantes, record y numeroVidasIniciales

**JuegoAdivinarNumero.java**

El código define una clase llamada JuegoAdivinaNumero que extiende la clase Juego y proporciona una implementación concreta del método jugar() de esa clase. La clase JuegoAdivinaNumero representa un juego en el que el objetivo del jugador es adivinar un número aleatorio.

La clase JuegoAdivinaNumero tiene los siguientes atributos:

* numeroAdivinar: Un objeto de tipo Integer que representa el número que el jugador debe adivinar.

La clase JuegoAdivinaNumero tiene los siguientes métodos:

* JuegoAdivinaNumero(Integer numeroVidasIniciales): Constructor de la clase que recibe como parámetro el número de vidas iniciales del jugador. Este constructor genera un número aleatorio entre 1 y 10 (ambos inclusive) que el jugador debe adivinar, y lo asigna al atributo numeroAdivinar.
* jugar(): Método que implementa la lógica principal del juego. En este caso, el jugador tiene que introducir un número entre 1 y 10 (ambos inclusive) usando el método UtilidadesJuego.leerNumero() de la clase UtilidadesJuego, y luego se compara con el número a adivinar. Si el número introducido es igual al número a adivinar, se muestra un mensaje de acierto y se actualiza el récord. Si el número introducido es incorrecto, se muestra una pista indicando si es mayor o menor que el número a adivinar, y se resta una vida al jugador utilizando el método quitaVida() de la clase Juego. Si el jugador se queda sin vidas, el juego se reinicia llamando de nuevo al método jugar().
* muestraPista(Integer numeroIntroducido): Método que muestra una pista al jugador indicando si el número introducido es mayor o menor que el número a adivinar.
* numeroValido(Integer numero): Método que verifica si un número introducido por el jugador es válido, es decir, si está entre 1 y 10 (ambos inclusive).
* muestraNombre(): Método que muestra el nombre del juego, en este caso "Adivina el número!!", implementando el método abstracto muestraNombre() de la clase Juego.
* muestraInfo(): Método que muestra información sobre el juego, en este caso "Introduce un número entre 1 y 10", implementando el método abstracto muestraInfo() de la clase Juego.

**JuegoAdivinarPar.java**

El código que proporcionas es una definición de una clase llamada JuegoAdivinaPar que extiende la clase JuegoAdivinaNumero y proporciona una implementación concreta de algunos métodos específicos del juego de adivinar números pares.

* La clase JuegoAdivinaPar extiende la clase JuegoAdivinaNumero, lo que significa que hereda todos los atributos y métodos de la clase JuegoAdivinaNumero. Esto indica que el juego de adivinar números pares es una variante del juego de adivinar números en general, con algunas reglas específicas para números pares.
* La clase tiene un constructor con un parámetro numeroVidasIniciales, que llama al constructor de la clase padre JuegoAdivinaNumero pasando el mismo parámetro, y además utiliza el método numeroParAleatorio() de la clase UtilidadesJuego para generar un número par aleatorio que será el número a adivinar en el juego.
* La clase sobrescribe el método numeroValido(Integer numero) de la clase padre JuegoAdivinaNumero. Este método verifica si el número introducido por el jugador es válido, es decir, si es mayor que 0, menor o igual a 10 (los límites establecidos en la clase padre) y si es un número par. Para ello, utiliza el método esPar(numero) de la clase UtilidadesJuego que devuelve true si el número es par, y si todas las condiciones se cumplen, retorna true, indicando que el número es válido. Si alguna de las condiciones no se cumple, imprime un mensaje de error y retorna false.
* La clase sobrescribe los métodos muestraNombre() y muestraInfo() de la clase padre JuegoAdivinaNumero. Estos métodos son utilizados para mostrar mensajes al jugador sobre el nombre del juego y las instrucciones para jugar. En este caso, los mensajes son personalizados para el juego de adivinar números pares.

**JuegoAdivinarImpar.java**

El código que proporcionas es una definición de una clase llamada JuegoAdivinaImpar que extiende la clase JuegoAdivinaNumero y proporciona una implementación concreta de algunos métodos específicos del juego de adivinar números impares.

* La clase JuegoAdivinaImpar extiende la clase JuegoAdivinaNumero, lo que significa que hereda todos los atributos y métodos de la clase JuegoAdivinaNumero. Esto indica que el juego de adivinar números impares es una variante del juego de adivinar números en general, con algunas reglas específicas para números impares.
* La clase tiene un constructor con un parámetro numeroVidasIniciales, que llama al constructor de la clase padre JuegoAdivinaNumero pasando el mismo parámetro, y además utiliza el método numeroImparAleatorio() de la clase UtilidadesJuego para generar un número impar aleatorio que será el número a adivinar en el juego.
* La clase sobrescribe el método numeroValido(Integer numero) de la clase padre JuegoAdivinaNumero. Este método verifica si el número introducido por el jugador es válido, es decir, si es mayor que 0, menor o igual a 10 (los límites establecidos en la clase padre) y si es un número impar. Para ello, utiliza el método esImpar(numero) de la clase UtilidadesJuego que devuelve true si el número es impar, y si todas las condiciones se cumplen, retorna true, indicando que el número es válido. Si alguna de las condiciones no se cumple, imprime un mensaje de error y retorna false.
* La clase sobrescribe los métodos muestraNombre() y muestraInfo() de la clase padre JuegoAdivinaNumero. Estos métodos son utilizados para mostrar mensajes al jugador sobre el nombre del juego y las instrucciones para jugar. En este caso, los mensajes son personalizados para el juego de adivinar números impares.

**JuegoAhorcado.java**

La clase JuegoAhorcado extiende de una clase base llamada Juego y proporciona la lógica para jugar al juego del ahorcado.

A continuación, se describen los elementos principales del código:

Declaración de variables de instancia:

* palabraAdivinar: una cadena de caracteres que representa la palabra que se debe adivinar en el juego del ahorcado.
* palabraAdivinarOculta: una cadena de caracteres que representa la palabra oculta que se muestra al jugador, con las letras no adivinadas reemplazadas por guiones ("-").
* letrasIntroducidas: una lista de cadenas de caracteres que representa las letras que el jugador ha introducido.

Constructor:

* JuegoAhorcado(String palabraAdivinar): es el constructor de la clase. Recibe como parámetro la palabra que se debe adivinar en el juego y la asigna a la variable palabraAdivinar. También llama al constructor de la clase base Juego pasándole el número máximo de vidas del jugador, que en este caso se establece en 7.

Métodos públicos:

* iniciarJuego(): inicializa la palabra oculta palabraAdivinarOculta con guiones ("-") si aún no ha sido inicializada.
* introducirLetra(): solicita al jugador que introduzca una letra y la devuelve como resultado después de validarla.
* comprobarLetraIntroducidaEnPalabraAdivinar(StringBuilder sbPalabraClave, String letraIntroducida): comprueba si la letra introducida por el jugador está presente en la palabra a adivinar. Si es así, actualiza la palabra oculta con la letra adivinada y muestra un mensaje de acierto. Si no es así, muestra un mensaje de error.
* comprobarResultadoJuego(boolean acierto): comprueba si el juego ha finalizado (si se ha adivinado la palabra o si el jugador ha perdido todas las vidas), y muestra el resultado correspondiente.
* letraValida(String cadenaIntroducida): comprueba si la letra introducida por el jugador es válida (no ha sido introducida antes, tiene longitud 1 y no es un espacio en blanco).
* muestraNombre(): muestra el nombre del juego, en este caso "El ahorcado!".
* muestraInfo(): muestra información sobre el estado actual del juego, incluyendo la palabra oculta y solicita al jugador que introduzca una letra.
* mostrarLetrasIntroducidas(): muestra las letras introducidas por el jugador hasta el momento.
* pintarAhorcado(): muestra en consola una representación gráfica del ahorcado, dependiendo del número de vidas restantes del jugador.
* jugar(): es el método principal del juego que inicia la lógica para jugar. Comprueba si la letra introducida por el jugador es correcta o no, actualiza la palabra oculta y muestra el resultado del juego.

Lógica método jugar:

* Se declara una variable booleana acierto y se inicializa con el valor false. Esta variable se utilizará para indicar si el jugador acertó la letra introducida o no.
* Se llama al método mostrarLetrasIntroducidas(), que muestra por pantalla las letras introducidas por el jugador hasta el momento.
* Se llama al método iniciarJuego(), que se encarga de inicializar el juego si es la primera vez que se juega. Este método muestra por pantalla el nombre del juego (en este caso "El Ahorcado") y oculta la palabra a adivinar reemplazando todas las letras por guiones bajos ("\_") en la variable palabraAdivinarOculta.
* Se crea un objeto StringBuilder llamado sbPalabraClave y se le asigna el valor de palabraAdivinarOculta. Este objeto se utilizará para llevar un registro de la palabra a adivinar oculta con las letras adivinadas por el jugador.
* Se llama al método introducirLetra(), que permite al jugador introducir una letra. El valor de la letra introducida se guarda en la variable letraIntroducida.
* Se llama al método comprobarLetraIntroducidaEnPalabraAdivinar(), que se encarga de verificar si la letra introducida por el jugador está presente en la palabra a adivinar. El resultado de esta comprobación se guarda en la variable acierto.
* Se llama al método comprobarResultadoJuego(), que se encarga de determinar el resultado del juego en función de si el jugador acertó la letra introducida o no.
* El turno de juego termina y el control se devuelve al flujo principal del programa.