Estado	Finalizado
Comenzado	martes, 4 de julio de 2023, 18:23
Completado	martes, 4 de julio de 2023, 20:45
Duración	2 horas 21 minutos
Calificación	**31** de 100
Comentario -	No aprobado

Pregunta 1

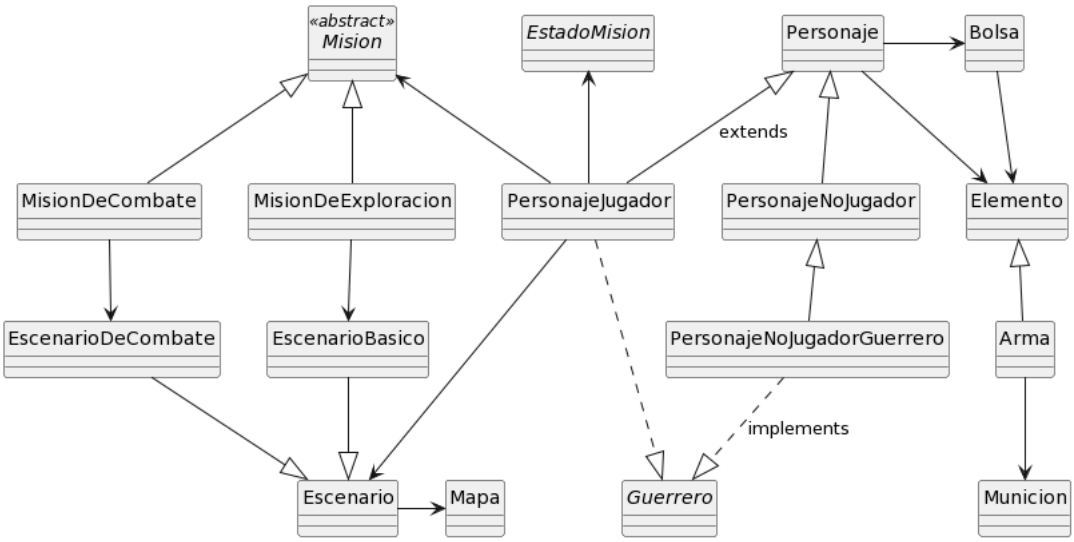
Sin contestar

Sin calificar

Esta actividad consiste en desarrollar los métodos del proyecto **`JuegoEscenarios`** indicados. El resto de las clases se encuentra lista para usar y no deben modificarlas. Las clases pedidas deben entregarse a través de las siguientes preguntas.

JAVA API

Descargar el Proyecto



Pregunta 2

Parcialmente correcta

Se puntúa 12 sobre 20

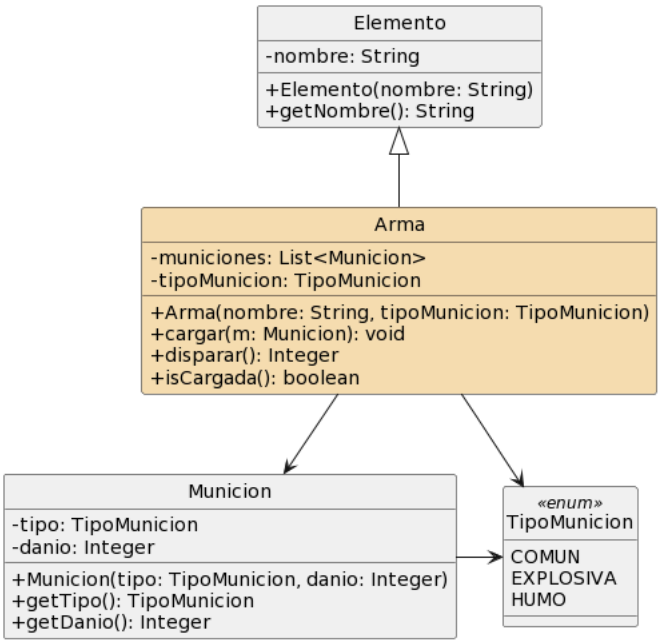
ATENCIÓN

1. El botón "**Precheck**" prueba el programa para el ejemplo. Lo pueden utilizar todas las veces que necesiten.
2. El botón "**Comprobar**" aplica todas las pruebas. Tienen tres usos sin penalidad, a partir de la tercera se descontarán puntos en la nota final.
3. TODO debe estar dentro de las funciones predefinidas.

ENUNCIADO

Implemente los métodos faltantes de la clase **Arma** según la documentación de la misma.

Simbolos necesarios : ( ) { } [ ] ; = < > " '



Por ejemplo:

Prueba	Resultado

`try {`
`System.out.println("Precheck");`
`Arma a = new Arma("Arma Municion Explosiva", TipoMunicion.EXPLOSIVA);`
`System.out.println(a.isCargada());`
`a.cargar(new Municion(TipoMunicion.EXPLOSIVA,10));`
`System.out.println(a.isCargada());`
`a.disparar();`
`System.out.println(a.isCargada());`
`} catch (Exception ex) {`
`System.out.println("Excepcion no esperada: " + ex.getClass().getSimpleName());`
`System.out.println("Mensaje: " + ex.getMessage());`
`}`
	`Precheck`
	`false`
	`true`
	`false`

Respuesta: (sistema de penalización: 0, 0, 0, 5, 10, 15, 20, 25 %)

Reiniciar respuesta

```
1  public class Arma extends Elemento
2  {
3      List<Municion> municiones;
4      TipoMunicion tipoMunicion;
5
6      /**
7       * Constructor de Arma
8       * @param nombre
9       * @param tipoMunicion
10      */
11     public Arma(String nombre, TipoMunicion tipoMunicion)
12     {
13         super(nombre);
14         municiones = new ArrayList<>();
15         // TODO - Implementar el constructor
16     }
```

```
    }

    /**
     * Agrega la municion a la lista de municiones.
     * Lanza MunicionNoValidaException si la municion no coincide con el tipo
     * de municion del arma.
     * @param m la municion a cargar
     * @throws MunicionNoValidaException si la municion no coincide con el
     *           tipo de municion del arma.
     */
    public void cargar (Municion m) throws MunicionNoValidaException
    {
        if((m.getTipo()!=TipoMunicion.COMUN)&&(m.getTipo()!=TipoMunicion.EXPLOSIVA)&&(m.getTipo()!=TipoMun
        {
            throw new MunicionNoValidaException("");
        }
        else
        {
            municiones.add(m);
        }
        // TODO - Implementar el metodo
    }

    /**
     * Remueve la primera municion de la lista y retorna el danio
     * de la misma.
     * Arroja ArmaDescargadaException si no hay municiones
     *
     * @return el valor de danio de la municion removida
     * @throws ArmaDescargadaException si no hay municiones disponibles
     */
    public Integer disparar() throws ArmaDescargadaException
    {
        Municion m = municiones.get(0);
        Integer danio = m.getDanio();
        if(municiones.size()==0)
```

	Prueba	Esperado	Conseguido	

✓	try {
	System.out.println("Precheck");
	Arma a = new Arma("Arma Municion Explosiva", TipoMunicion.EXPLOSIVA);
	System.out.println(a.isCargada());
	a.cargar(new Municion(TipoMunicion.EXPLOSIVA,10));
	System.out.println(a.isCargada());
	a.disparar();
	System.out.println(a.isCargada());
	} catch (Exception ex) {
	System.out.println("Excepcion no esperada: " + ex.getClass().getSimpleName());
	System.out.println("Mensaje: " + ex.getMessage());
	}
		Precheck
		false
		true
		false
			Precheck
			false
			true
			false
				✓

Tu programa ha fallado en uno o más de las pruebas ocultas

Parcialmente correcta

Puntos para este envío: 12/20.

Pregunta 3

Parcialmente correcta

Se puntúa 8 sobre 20

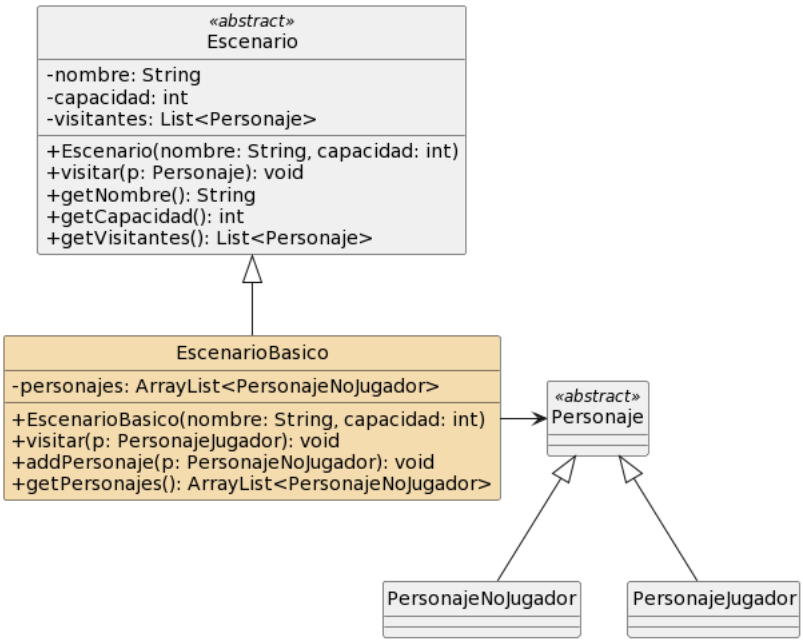
ATENCIÓN

1. El botón "**Precheck**" prueba el programa para el ejemplo. Lo pueden utilizar todas las veces que necesiten.
2. El botón "**Comprobar**" aplica todas las pruebas. Tienen tres usos sin penalidad, a partir de la tercera se descontarán puntos en la nota final.
3. TODO debe estar dentro de las funciones predefinidas.

ENUNCIADO

Implemente los métodos faltantes de la clase **EscenarioBasico** según la documentación de la misma.

Simbolos necesarios : ( ) { } [ ] ; = < > " '



Por ejemplo:

Prueba	Resultado

`try {`
`System.out.println("Precheck");`
`EscenarioBasico e = new EscenarioBasico("Value Stop", 2);`
`e.addPersonaje(new PersonajeNoJugador("Mr. Roper", 10 , null, null));`
`PersonajeJugador p = new PersonajeJugador("Ash Williams",100);`
`System.out.println(p.getEscenarioActual());`
`e.visitar(p);`
`System.out.println(p.getEscenarioActual().getNombre());`
`System.out.println(e.getVisitantes().get(0).getNombre());`
`} catch (Exception ex) {`
`System.out.println("Excepcion no esperada: " + ex.getClass().getSimpleName());`
`System.out.println("Mensaje: " + ex.getMessage());`
`}`
	`Precheck`
	`null`
	`Value Stop`
	`Ash Williams`

Respuesta: (sistema de penalización: 0, 0, 0, 5, 10, 15, 20, 25 %)

Reiniciar respuesta

```
1  public class EscenarioBasico extends Escenario
2  {
3      private ArrayList<PersonajeNoJugador> personajes;
4
5      /**
6       * Crea un escenario basico con nombre y capacidad de alojar
7       * personajes no jugadores.
8       *
9       * @param nombre El nombre del escenario
10      * @param capacidad La cantidad de Personajes no jugadores que pueden
11      *                  agregarse
12      */
13     public EscenarioBasico(String nombre, Integer capacidad)
14     {
```

```
        super(nombre,capacidad);
        personajes = new ArrayList<PersonajeNoJugador>();
        // TODO - Implementar el constructor
    }

    /**
     * Visita el escenario actualizando el escenarioActual del personaje
     * y agregandolo a la lista de visitantes.
     *
     * @param p el personaje jugador que visita el escenario
     * @throws IllegalStateException si el personaje jugador tiene un arma como elemento actual
     */
    @Override
    public void visitar(PersonajeJugador p)
    {
        // TODO - Implementar el metodo
    }

    /**
     * Agrega un personaje no jugador al escenario.
     *
     * @param p el personaje no jugador a agregar
     * @throws IllegalStateException si el escenario esta lleno y no puede agregar mas personajes
     */
    public void addPersonaje(PersonajeNoJugador p)
    {
        if(this.getCapacidad() <= personajes.size())
        {
            throw new IllegalStateException();
        }
        personajes.add(p);
        // TODO - Implementar el metodo
    }

    /**
     * Devuelve una lista de PersonajeNoJugador que representa los personajes
     * no jugadores actualmente en el escenario.
     *
```

	Prueba	Esperado	Conseguido	

⊗	try {
	System.out.println("Precheck");
	EscenarioBasico e = new EscenarioBasico("Value Stop", 2);
	e.addPersonaje(new PersonajeNoJugador("Mr. Roper", 10 , null, null));
	PersonajeJugador p = new PersonajeJugador("Ash Williams",100);
	System.out.println(p.getEscenarioActual());
	e.visitar(p);
	System.out.println(p.getEscenarioActual().getNombre());
	System.out.println(e.getVisitantes().get(0).getNombre());
	} catch (Exception ex) {
	System.out.println("Excepcion no esperada: " +
	ex.getClass().getSimpleName());
	System.out.println("Mensaje: " + ex.getMessage());
	}
		Precheck
		null
		Value Stop
		Ash
		Williams
			Precheck
			null
			Excepcion no esperada:
			NullPointerException
			Mensaje: null
				⊗

También se han fallado algunas pruebas ocultas.

Mostrar diferencias

Parcialmente correcta

Puntos para este envío: 8/20.

Pregunta 4

Parcialmente correcta

Se puntúa 7 sobre 30

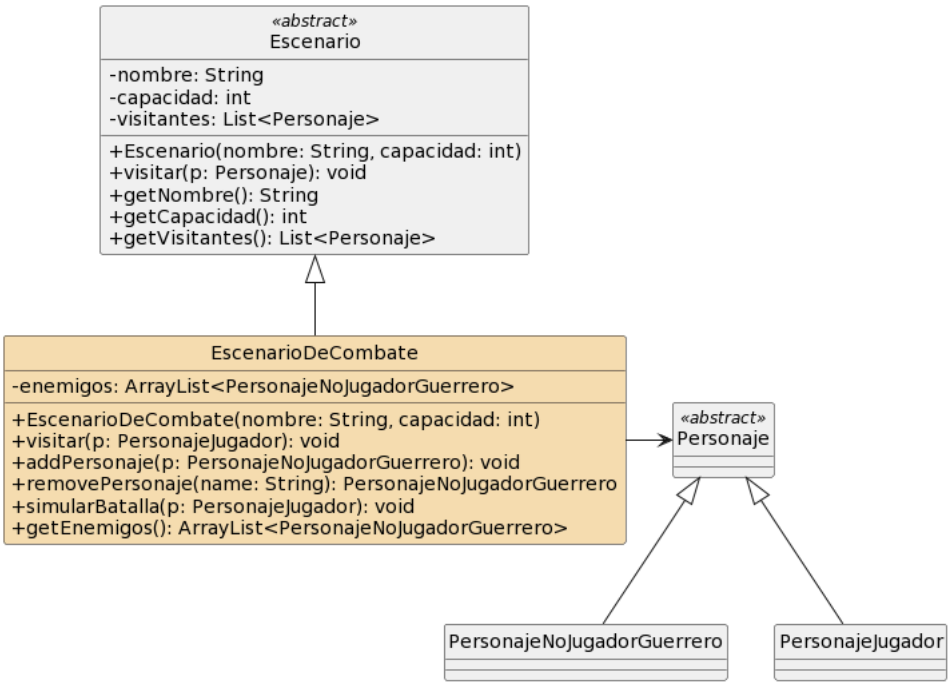
ATENCIÓN

1. El botón "**Precheck**" prueba el programa para el ejemplo. Lo pueden utilizar todas las veces que necesiten.
2. El botón "**Comprobar**" aplica todas las pruebas. Tienen tres usos sin penalidad, a partir de la tercera se descontarán puntos en la nota final.
3. TODO debe estar dentro de las funciones predefinidas.

ENUNCIADO

Implemente los métodos faltantes de la clase **EscenarioDeCombate** según la documentación de la misma.

Simbolos necesarios : ( ) { } [ ] ; = < > " '



Por ejemplo:

Prueba	Resultado

`try {`
`System.out.println("Precheck");`
`EscenarioDeCombate e = new EscenarioDeCombate("Arena1", 1);`
`Arma arma1 = new Arma("Arco", TipoMunicion.COMUN);`
`arma1.cargar(new Municion(TipoMunicion.COMUN, 5));`
`PersonajeNoJugadorGuerrero enemigo = new PersonajeNoJugadorGuerrero("Evil Dead", 100, arma1);`
`Arma arma2 = new Arma("Arco Fuego", TipoMunicion.EXPLOSIVA);`
`arma2.cargar(new Municion(TipoMunicion.EXPLOSIVA,50));`
`arma2.cargar(new Municion(TipoMunicion.EXPLOSIVA,50));`
`PersonajeJugador jugador = new PersonajeJugador("Ash", 100);`
`jugador.setElementoActual(arma2);`
`e.addPersonaje(enemigo);`
`System.out.println("Enemigos: " +`
`e.getEnemigos().stream().map(Personaje::getNombre).collect(Collectors.toList()));`
`System.out.println("Vida Jugador: " + jugador.getVida());`
`e.visitar(jugador);`
`System.out.println("Jugador escenario actual: " + jugador.getEscenarioActual().getNombre());`
`System.out.println("Visitante: " + e.getVisitantes().get(0).getNombre());`
`System.out.println("Enemigos: " +`
`e.getEnemigos().stream().map(Personaje::getNombre).collect(Collectors.toList()));`
`System.out.println("Vida Jugador: " + jugador.getVida());`
`} catch (Exception e) {`
`System.out.println("Excepcion no esperada: " + e.getClass().getSimpleName());`
`System.out.println("Mensaje: " + e.getMessage());`
`}`
	`Precheck`
	`Enemigos: [Evil Dead]`
	`Vida Jugador: 100`
	`Jugador escenario actual:`
	`Arena1`
	`Visitante: Ash`
	`Enemigos: []`
	`Vida Jugador: 100`

Respuesta: (sistema de penalización: 0, 0, 0, 5, 10, 15, 20, 25 %)

Reiniciar respuesta

```
public class EscenarioDeCombate extends Escenario
{
    private ArrayList<PersonajeNoJugadorGuerrero> enemigos;

    public EscenarioDeCombate(String nombre, Integer capacidad)
    {
        super(nombre, capacidad);
        enemigos = new ArrayList<>();
    }

    /**
     * Actualiza la posicion actual del personaje y agrega el personaje
     * a la lista de visitantes del escenario.
     * Si hay enemigos y el personaje tiene un arma como elemento actual
     * simula ademas una batalla (metodo interno)
     * Si el numero de enemigos en el escenario es mayor a cero y el
     * personaje no tiene como elemento actual un arma lanza
     * IllegalStateException y no actualiza la lista de visitantes ni la
     * posicion del personaje.
     * @param p El pesonaje jugador que visita el escenario
     * @throws IllegalStateException cuando hay enemigos y el personaje
     * no tiene seleccionada un arma
     */
    @Override
    public void visitar(PersonajeJugador p)
    {
        // TODO - Implementar el metodo
    }

    /**
     * Agrega un personaje a la lista de personajes enemigos del escenario.
     * Debe validar el nombre para evitar duplicados. En caso de que ya
     * exista un personaje con el mismo nombre arroja InvalidArgumentException
     *
     * @param p el personaje no jugador guerrero a agregar
     * @throws IllegalArgumentException si hay un personaje con el mismo
     * nombre en la lista
     */
    public void addPersonaje(PersonajeNoJugadorGuerrero p)
    {
        if(enemigos.contains(p.getNombre()))
        {
            throw new IllegalArgumentException();
        }
        else
        {
            enemigos.add(p);
        }
        // TODO - Implementar el metodo
    }

    /**
```

	Prueba	Esperado	Conseguido	

⊗	try {
	System.out.println("Precheck");
	EscenarioDeCombate e = new EscenarioDeCombate("Arena1", 1);
	Arma arma1 = new Arma("Arco", TipoMunicion.COMUN);
	arma1.cargar(new Municion(TipoMunicion.COMUN, 5));
	PersonajeNoJugadorGuerrero enemigo = new PersonajeNoJugadorGuerrero("Evil Dead", 100, arma1);
	Arma arma2 = new Arma("Arco Fuego", TipoMunicion.EXPLOSIVA);
	arma2.cargar(new Municion(TipoMunicion.EXPLOSIVA,50));
	arma2.cargar(new Municion(TipoMunicion.EXPLOSIVA,50));
	PersonajeJugador jugador = new PersonajeJugador("Ash", 100);
	jugador.setElementoActual(arma2);
	e.addPersonaje(enemigo);
	System.out.println("Enemigos: " +
	e.getEnemigos().stream().map(Personaje::getNombre).collect(Collectors.toList()));
	System.out.println("Vida Jugador: " + jugador.getVida());
	e.visitar(jugador);
	System.out.println("Jugador escenario actual: " +
	jugador.getEscenarioActual().getNombre());
	System.out.println("Visitante: " + e.getVisitantes().get(0).getNombre());
	System.out.println("Enemigos: " +
	e.getEnemigos().stream().map(Personaje::getNombre).collect(Collectors.toList()));
	System.out.println("Vida Jugador: " + jugador.getVida());
	} catch (Exception e) {
	System.out.println("Excepcion no esperada: " + e.getClass().getSimpleName());
	System.out.println("Mensaje: " + e.getMessage());
	}
		Precheck
		Enemigos: [Evil Dead]
		Vida Jugador: 100
		Jugador escenario actual: Arena1
		Visitante: Ash
		Enemigos: []
		Vida Jugador: 100
			Precheck
			Enemigos: [Evil Dead]
			Vida Jugador: 100
			Excepcion no esperada: NullPointerException
			Mensaje: null
				⊗

También se han fallado algunas pruebas ocultas.

Mostrar diferencias

Parcialmente correcta

Puntos para este envío: 7/30.

Pregunta 5

Parcialmente correcta

Se puntúa 5 sobre 30

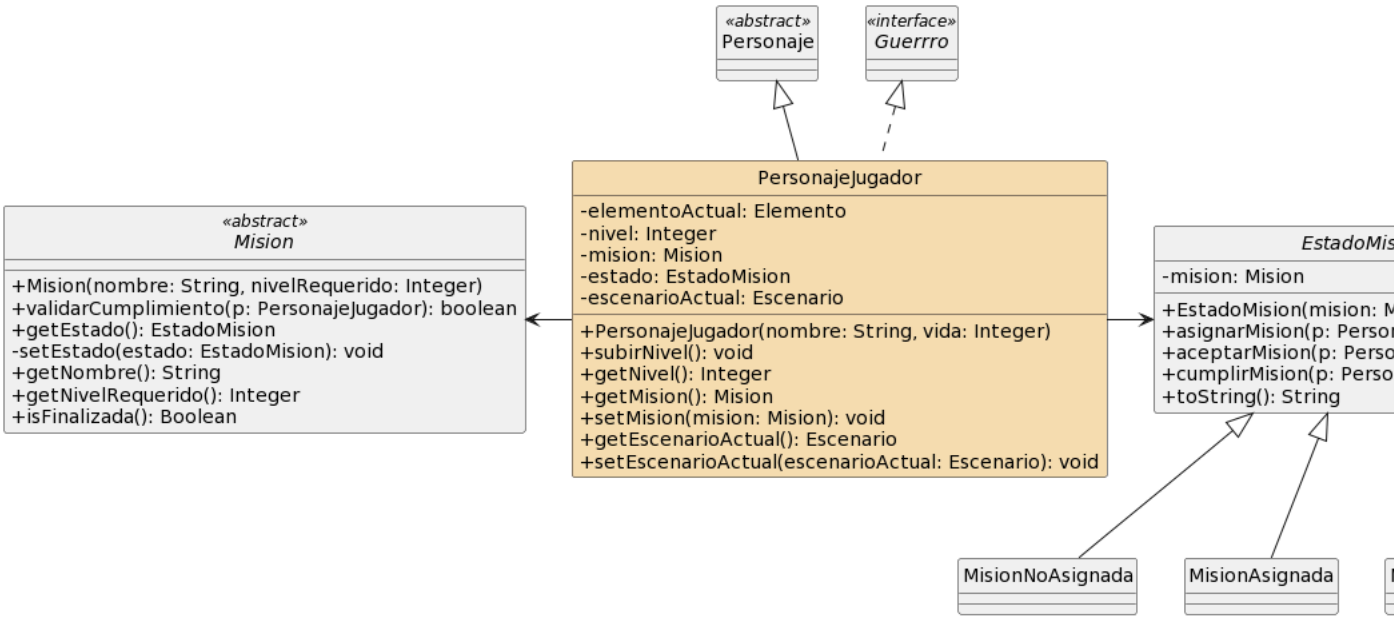
ATENCIÓN

1. El botón "**Precheck**" prueba el programa para el ejemplo. Lo pueden utilizar todas las veces que necesiten.
2. El botón "**Comprobar**" aplica todas las pruebas. Tienen tres usos sin penalidad, a partir de la tercera se descontarán puntos en la nota final.
3. TODO debe estar dentro de las funciones predefinidas.

ENUNCIADO

Implemente los métodos faltantes de la clase **PersonajeJugador** según la documentación de la misma.

Simbolos necesarios : ( ) { } [ ] ; = < > " '



Por ejemplo:

Prueba	Resultado

`try {`
`System.out.println("Precheck");`
`EscenarioBasico escenario = new EscenarioBasico("Hell", 2);`
`Mision m = new MisionDeExploracion("La mision", 1, escenario);`
`PersonajeJugador personaje = new PersonajeJugador("Ruby Knowby", 200);`
`personaje.asignarMision(m);`
`System.out.println(personaje.getEstadoMision());`
`personaje.aceptarMision();`
`System.out.println(personaje.getEstadoMision());`
`escenario.visitar(personaje);`
`personaje.cumplirMision();`
`System.out.println(personaje.getEstadoMision());`
`} catch (Exception ex) {`
`System.out.println("Excepcion no esperada: " + ex.getClass().getSimpleName());`
`System.out.println("Mensaje: " + ex.getMessage());`
`}`
	`Precheck`
	`MisionAsignada`
	`MisionAceptada`
	`MisionFinalizada`

Respuesta: (sistema de penalización: 0, 0, 0, 5, 10, 15, 20, 25 %)

Reiniciar respuesta

```
1  class PersonajeJugador extends Personaje implements Guerrero {
2      private Elemento elementoActual;
3      private Integer nivel;
4      private Mision mision;
5      private Escenario escenarioActual;
6      private EstadoMision estado;
7      /**
8       * Crea un nuevo Personaje Jugador con su nombre y cantidad de vida
9       *
10      * @param nombre El nombre del personaje
11      * @param vida La vida del personaje
12      */
```

```
13    public PersonajeJugador(String nombre, Integer vida) {
14        super(nombre, vida);
15        this.nivel = 1;
16        this.estado = new MisionNoAsignada();
17    }
18
19    /**
20     * Retorna el elemento actual del personaje jugador.
21     *
22     * @return el elemento actual del personaje jugador
23     */
24    public Elemento getElementoActual() {
25        return elementoActual;
26    }
27
28    /**
29     * Establece el elemento actual del personaje jugador.
30     *
31     * @param elementoActual el nuevo elemento actual del personaje jugador
32     */
33    public void setElementoActual(Elemento elementoActual) {
34        this.elementoActual = elementoActual;
35    }
36
37    /**
38     * Realiza un ataque al enemigo.
39     *
40     * @param enemigo el personaje enemigo al que se va a atacar
41     * @throws ArmaDescargadaException si el arma estÃ¡ descargada y no se puede disparar
42     */
43    @Override
44    public void atacar(Personaje enemigo) throws ArmaDescargadaException {
45        enemigo.recibirDanio(((Arma) elementoActual).disparar());
46    }
47
48    /**
49     * Guarda el elemento actual en la bolsa.
50     *
51     * @throws BolsaLlenaException si la bolsa estÃ¡ llena y no se puede guardar el elemento
52     */
```

	Prueba	Esperado	Conseguido	

⊗	try {
	System.out.println("Precheck");
	EscenarioBasico escenario = new EscenarioBasico("Hell", 2);
	Mision m = new MisionDeExploracion("La mision", 1, escenario);
	PersonajeJugador personaje = new PersonajeJugador("Ruby Knowby",
	200);
	personaje.asignarMision(m);
	System.out.println(personaje.getEstadoMision());
	personaje.aceptarMision();
	System.out.println(personaje.getEstadoMision());
	escenario.visitar(personaje);
	personaje.cumplirMision();
	System.out.println(personaje.getEstadoMision());
	} catch (Exception ex) {
	System.out.println("Excepcion no esperada: " +
	ex.getClass().getSimpleName());
	System.out.println("Mensaje: " + ex.getMessage());
	}
		Precheck
		MisionAsignada
		MisionAceptada
		MisionFinalizada
			Precheck
			MisionNoAsignada
			Excepcion no esperada:
			IllegalStateException
			Mensaje: null
				⊗

También se han fallado algunas pruebas ocultas.

Mostrar diferencias

Parcialmente correcta

Puntos para este envío: 5/30. Contando con los intentos anteriores, daría **5/30**.