**GUÍA DE APRENDIZAJE**

**Herencia**

|  |  |
| --- | --- |
| **OBJETIVO DE APRENDIZAJE** | **OA 4**. Construir aplicaciones computacionales basadas en programación orientada a objetos, de manera de cumplir con las exigencias técnicas y de los usuarios. |
| **APRENDIZAJE ESPERADO** | **AE 5.** Construye programas, utilizando los componentes del paradigma orientado a objetos, a partir del análisis de problemáticas y requerimientos, modelando el comportamiento del sistema.  **AE 1.** Construye unidades de prueba para verificar el correcto funcionamiento de la codificación realizada, de acuerdo a exigencias técnicas de confiabilidad.  **AE 2.** Detecta y corrige errores de codificación, analizando el comportamiento del código de programación, de acuerdo a especificaciones y manuales de referencia. |
| **CRITERIOS DE EVALUACIÓN** | **5.1.** Construye los componentes a utilizar en el programa a partir del análisis de requerimientos para resolución de problema, resguardando los principios de POO.  **5.2.** Desarrolla interfaz de usuario simple, basado en programación orientada a objetos, para navegar a través del programa, considerando requerimientos circunstanciales de un determinado problema.  **1.1.** Realiza pruebas para detectar problemas previos a la codificación de una unidad de software con una herramienta de software disponible en el mercado.  **1.2.** Construye una unidad de prueba para detectar excepciones en un entorno de lenguaje con una herramienta compatible que permita desarrollar, configurar e Implementar aplicaciones en función del lenguaje utilizado.  **1.3.** Construye una unidad de prueba para detectar casos límites, en un entorno de lenguaje, con una herramienta compatible, propia del software utilizado como marco de trabajo.  **2.1.** Utiliza la herramienta de depuración de un entorno de desarrollo, para revisar la codificación de clases programadas.  **2.2.** Revisa y corrige las variables, del programa desarrollado para la solución de un caso, indagando en las funciones internas de acuerdo a protocolos de revisión.  **2.3.** Revisa las variables de una función de clases programadas, utilizando opciones del lenguaje empleado, para asegurar su correcto funcionamiento. |
| **OBJETIVOS DE APRENDIZAJE GENÉRICOS** | **A – B – C – G - H** |

**Herencia**

La herencia es uno de los pilares de la programación orientada a objetos y es el mecanismo por el cual una clase hereda los atributos y métodos de otra clase. También es considerada una herramienta para la reutilización de código, ya que al momento de crear una clase que contiene elementos de otra ya creada, podemos utilizarla para evitar escribir el código nuevamente.

En Java, la clase que se hereda se denomina **superclase**, mientras que la que recibe los atributos y métodos, es la **subclase**. Por lo tanto, la subclase es una especialización de la superclase.

A continuación crearemos un [Battle Royale](https://www.gamerdic.es/termino/battle-royale/) (muy sencillo por lo demás), comenzando con un jugador básico:

|  |
| --- |
| public class Jugador {   private String nombre;  private int hp; *//vida del jugador*  private int mp; *//magia del jugador*   public Jugador(String nombre, int hp, int mp) {  this.nombre = nombre;  this.hp = hp;  this.mp = mp;  }   public String getNombre() {  return nombre;  }  public void setNombre(String nombre) {  this.nombre = nombre;  }  public int getHp() {  return hp;  }  public void setHp(int hp) {  this.hp = hp;  }  public int getMp() {  return mp;  }  public void setMp(int mp) {  this.mp = mp;  }  public void correr()  {  System.out.println("El jugador corre");  }  public void decirNombre()  {  System.out.println("Soy " + nombre + " ¡muahahaha!");  }  public void ataqueBasico(Jugador objetivo)  {  objetivo.hp -= 10;  }  public void recibirDaño (int daño)  {  this.hp -= daño;  }    void mensajeSecreto()  {  System.out.println("Este es un mensaje secreto de Jugador");  } } |

Ahora para crear un personaje “Mago”, no es necesario crear los mismos atributos y métodos, simplemente usamos la palabra reservada ***extends***para heredar los comportamientos de la clase “Jugador”, pero además incorporando nuevas funcionalidades, en este caso dos poderes, como “bloquear camino” y “bola de fuego”.

|  |
| --- |
| public class Mago extends Jugador {  String fraseDeMago;   *//Constructor*  public Mago (String nombre, int hp, int mp, String fraseDeMago)  {  super(nombre, hp, mp);  this.fraseDeMago = fraseDeMago;  }   *//Poderes*  public void bloquearCamino(Jugador objetivo)  {  int daño = 30;  int gastoMp = 20;  System.out.println("Youuu shall not paass!!");  System.out.println(getNombre() + " bloqueó camino a " + objetivo.getNombre());  objetivo.recibirDaño(daño);  setMp(getMp()- gastoMp);  }   public void fireball (Jugador objetivo)  {  int daño = 60;  System.out.println("Bolaa de fuegooo");  System.out.println(getNombre() + " tiró bola de fuego a " + objetivo.getNombre());  objetivo.recibirDaño(daño);  *//Reducir magia*  }  public void mensajeSecreto()  {  System.out.println("Este es un mensaje secreto de Mago");  }  public void llamar()  {  this.mensajeSecreto(); *//accede a los métodos y atributos de la clase actual.*  super.mensajeSecreto(); *//accede a los métodos y atributos de la clase padre.*  } } |

Se puede apreciar que en la jerarquía, tanto las superclases como las subclases tienen sus propios constructores. Usando la palabra reservada ***super***,estamos referenciando a los miembros de la superclase. Para un mayor detalle se adjuntan los constructores de ambas clases.

|  |
| --- |
| public Jugador(String nombre, int hp, int mp) {  this.nombre = nombre;  this.hp = hp;  this.mp = mp;  }   public Mago (String nombre, int hp, int mp, String fraseDeMago)  {  super(nombre, hp, mp);  this.fraseDeMago = fraseDeMago;  } |

Otro ejemplo del uso de super, se encuentra en la clase “Mago”. Este tiene un método llamar(), que invoca dos métodos con el mismo nombre (mensajeSecreto()), sin embargo, uno referencia al objeto de tipo “Mago”, y otro al objeto de la superclase “Jugador”.

¡Incluso es posible crear un MagoMaestro! siendo la evolución de un Mago tradicional:

|  |
| --- |
| public class MagoMaestro extends Mago {   public MagoMaestro(String nombre, int hp, int mp, String fraseDeMago) {  super(nombre, hp, mp, fraseDeMago);  }   public void superFireball(Jugador objetivo)  {  int daño = 80;  int gastoMp = 20;  System.out.println("¡¡Super bola de fuego!!");  System.out.println(getNombre() + " lanzó super bola de fuego a " + objetivo.getNombre());  objetivo.recibirDaño(daño);  setMp(getMp()- gastoMp);  } } |

**RETO**

|  |
| --- |
| **¡Ya trabajaste en los Retos anteriores!**  **Ahora es tu turno y aplica los pasos efectuados para el siguiente desafío.**  **Recuerda consultar con el/la docente en caso dudas o comentarios.** |

1. La Biblioteca Nacional te ha encargado modernizar el sistema de libros y revistas que posee en su inventario. Las consideraciones son las siguientes:
   1. Las características comunes de ambos objetos son:
      * Código
      * Título
      * Año de Publicación
   2. Los Libros además tienen el siguiente atributo:
      * ISBN (¡¿Qué es esto?! [Click aquí](https://www.isbn-international.org/es/content/%C2%BFqu%C3%A9-es-un-isbn))
   3. Por su parte, las revistas contienen:
      * Número de la revista
   4. Ambos objetos deben tener métodos getters, setters y toString().
   5. Debe implementar herencia.
   6. Además, debe implementar una interfaz de usuario que agregue los libros y revistas por teclado.
   7. **Bonus 1:** Los libros pueden ser prestados. Para ello, deberán agregar un atributo “prestado”. Cuando se crea el objeto, este no debe estar prestado.
      * Para prevenir posibles cambios en el programa, se tiene que implementar una interfaz “Prestable”, con los métodos prestar(), devolver() y prestado() (que retorna si el libro está prestado). La clase libro implementa esa Interfaz.
   8. **Bonus 2:** Notificar que el libro no puede ser prestado si ya lo está.
   9. Entregable:
      * Deberás subir a GitHub el código fuente con un README.txt que detalle la solución del programa realizado y un paso a paso para que el programa funcione correctamente.
      * Fecha de entrega: XX/XX/XXXX
   10. Consideraciones:
       * Buena documentación de código fuente a través de comentarios.
       * Realizar commit en Github por cada funcionalidad o ajuste realizado.
       * ¡Evitar subir el trabajo a Github de una vez!





**Recursos**

1. latincoder. Interfaces en Java: <https://www.youtube.com/watch?v=7MlB-K9AMxY&feature=emb_title>
2. Programación ATS. Curso completo de programación en Java: <https://www.youtube.com/watch?v=2ZXiuh0rg3M&list=PLWtYZ2ejMVJkjOuTCzIk61j7XKfpIR74K>
3. TutorialesYa. Curso completo de programación en Java: <http://www.tutorialesprogramacionya.com/javaya/>
4. La Geekipedia de Ernesto. Curso de Java desde cero: <https://www.youtube.com/watch?v=L1oMLsiMusQ&list=PLyvsggKtwbLX9LrDnl1-K6QtYo7m0yXWB>
5. Java desde cero: <https://javadesdecero.es/>
6. w3schools: <https://www.w3schools.com/java/default.asp>
7. CodingBat: <https://codingbat.com/java>
8. Learn Java: <https://www.learnjavaonline.org/>
9. CodeGym: <https://codegym.cc/>
10. Programming Skills:  
    <http://www.pskills.org/corejava.jsp>
11. Tutorials Point: <https://www.tutorialspoint.com/java/java_generics.htm>
12. Java Tests: <https://www.studytonight.com/tests/?subject=java>
13. JavatPoint: <https://www.javatpoint.com/core-java-quiz>
14. Recopilación de ejercicios Java POO: <https://www.academia.edu/36182478/Recopilaci%C3%B3n_de_Ejercicios_Java_POO_Programaci%C3%B3n_Orientada_a_Objetos>
15. Curso de programación orientada a objetos: <https://www.curriculumnacional.cl/614/w3-article-209545.html>
16. Curso de desarrollo de videojuegos: <https://www.curriculumnacional.cl/614/w3-article-209538.html>
17. Curso de desarrollo de aplicaciones móviles: <https://www.curriculumnacional.cl/614/w3-article-209536.html>
18. Makigas. Tutorial de GIT: <https://www.youtube.com/watch?v=jSJ8xhKtfP4&list=PLTd5ehIj0goMCnj6V5NdzSIHBgrIXckGU>

**Referencias**

1. MIT. (2010). MIT. (2010). Introduction to Programming in Java, Lecture 7: Inheritance, exceptions, file I/O. Recuperado 20 de agosto de 2020, de. <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/lecture-notes/MIT6_092IAP10_lec07.pdf>
2. UNAM. (n.f.). Guía práctica de estudio 07: Herencia. Recuperado 20 de Agosto de 2020, de <http://odin.fi-b.unam.mx/salac/practicasPOO/P05-POO-Encapsulamiento.pdf>.