**HOJA INTERACTIVA DE APRENDIZAJE Nº 03**

**Apellidos y Nombres:** Casas Moreno Christian Joel

**Docente:** Nestor Audante Ramos **Fecha de entrega:** 23/10/2021

**INTRODUCCIÓN**

1. **Indicadores:**
   1. Identifica la importancia de herencia.
   2. Identifica el uso de clases abstractas.
   3. Identifica el uso del constructor y métodos en subclases.
   4. Crea clases abstractas, superclases y subclases en la solución de un caso de estudio.
2. **Actividades para desarrollar en forma individual:** 
   1. Complete la tabla sobre la sintaxis Java:

|  |  |
| --- | --- |
| **Código fuente** | **Errores** |
| public class abstract T{  datos: String;  void abstract imprimir(){ }  } | -abstract debe ir antes de class.  -los datos se deben colocar como String datos;  -abstract debe ir antes de void.  -como es abstracta, las llaves no deben ir. |
| public class T{  datos: String;  abstract Integer marcar(String this.datos);  } | -los datos se deben colocar como String datos;  -se utiliza un dato como parámetro, el this no debe ir. |
| package com.tesla.\*;  abstract class T{  static abstract Integer marcar(String this.datos);  } | -El package no es el correcto.  -se utiliza un dato como parámetro, el this no debe ir.  -los métodos estáticos no pueden ser abstractos. |

* 1. Complete la tabla escribiendo los conceptos aprendidos en clase

|  |
| --- |
| **Conceptos** |
| **Constructor**  Método especial de una clase que se llama automáticamente siempre que se declara un objeto de esa clase. Su función es crear e inicializar un objeto creado a partir de una clase. |
| **Herencia**  La herencia permite que se puedan definir nuevas clases basadas de unas ya existentes a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación.  **Importancia de herencia**  La importancia de la herencia radica en que cuando una clase deriva de otra, esta hereda sus atributos y métodos y puede añadir nuevos atributos, métodos o redefinir los que han sido heredados. Es decir, la herencia permite compartir automáticamente métodos y datos entre clases, subclases y objetos. |
| **Clase abstracta**  Una clase que declara la existencia de métodos pero no la implementación de dichos métodos (o sea, las llaves { } y las sentencias entre ellas), se considera una clase abstracta.  Una clase abstracta puede contener métodos no abstractos, pero al menos uno de los métodos debe ser declarado abstracto.  Para declarar una clase o un método como abstractos, se utiliza la palabra reservada abstract.  **Uso de clases abstractas**  Una clase abstracta no se puede instanciar, pero si se puede heredar y las clases hijas serán las encargadas de agregar la funcionalidad a los métodos abstractos. Si no lo hacen así, las clases hijas deben ser también abstractas. |
| **Método abstracto**  Un método abstracto es un método declarado, pero no implementado, es decir, es un método del que solo se escribe su nombre, parámetros y tipo devuelto, pero no su código. |
| **Superclase**  Es la clase cuyas características (métodos, atributos, etc.) se heredan a las subclases. |
| **Subclase**  Una subclase (o una clase derivada, clase extendida o clase hija) es aquella que hereda características (métodos, atributos, etc.) de una superclase.  **¿Qué sucede con el constructor de una subclase?**  El constructor de una subclase solo funciona para esa subclase o se puede invocar desde una subclase de esta. Esta subclase no hereda el constructor de su superclase, pero lo puede invocar con super().  **¿Qué sucede si sobre escribo un método existente en la superclase?**  Si sobre escribe un método que ya existe en la superclase desde una subclase, pues el método de la subclase será el que tenga validez cuando sea llamado el método y el objeto sea de la subclase. |

* 1. Complete la tabla corrigiendo el código fuente.

|  |
| --- |
| **Código fuente** |
| class Objeto{  private Objeto(){  }  Objeto(int x){  }  }  class Temp extends Objeto  {    Temp()  {  this(5);  System.out.println("Constructor por defecto");  }  Temp(int x)  {  this(5, 15);  System.out.println(x);  }  Temp(int x, int y)  {  System.out.println(x \* y);  }  public abstract String mensaje(String args[])  {  Temp r = new Temp();  return “ok”;  }  } |

|  |
| --- |
| **Código fuente corregido** |
| class Objeto{  public Objeto(){   }  Objeto(int x){   } } class Temp extends Objeto {   private int x;  private int y;   Temp()  {  super();   System.*out*.println("Constructor por defecto");  }   Temp(int x)  {  System.*out*.println(x);  }   Temp(int x, int y)  {   System.*out*.println(x \* y);  }   public String mensaje(String args[])  {  Temp r = new Temp();  return "ok";  } } |

* 1. Caso de estudio:

Un negocio de transporte de frutas trabaja en la localidad de Urubamba y traslada frambuesas y arándanos hasta la ciudad de Cuzco mediante camiones de carga. Cada pedido que le realizan tiene información del cliente, cantidad de kilogramos por cada tipo de fruta, la dirección de entrega.

**Defina las clases necesarias para resolver los pedidos de transporte de fruta.**

El negocio genera muchas utilidades y decide ampliar los productos a trasladar. Ahora ofrece el traslado de maíz y alcachofas. Además del transporte terrestre ahora debe permitir el fluvial para atender los pedidos que llegan de Río Branco - Brasil.

**Defina las subclases clases necesarias para resolver los pedidos de transporte de fruta actuales.**

* 1. Suba el documento resuelto y la solución del caso de estudio en un directorio llamado lab-3.

1. **Referencia de material** 
   1. Equipo Geek (2019). ¿Qué es la Herencia en programación orientada a objetos?. Recuperado de <https://ifgeekthen.everis.com/es/herencia-en-programacion-orientada-objetos>

1. **Conclusiones y recomendaciones**