Parcial Programación, Parte Teórica

1. Dadas las siguientes clases:

Guerrero **extends** Personaje

Arquero **extends** Personaje **implements** Recargable  
Catapulta **implements** Recargable  
  
Indique si cada una de las siguientes sentencias es correcta, justificando en todos los casos:  
  
a) Guerrero conan = new Guerrero();  
b) Guerrero legolas = new Arquero();

c) Arquero legolas = new Catapulta();

d) Recargable artillero = new Catapulta();

e) Personaje conan = new Arquero();

1. Cuál será la salida por consola cuando el siguiente código sea compilado y ejecutado:  
     
   **class** C1 {

**public** C1() { System.out.print(“1”);  
**super**(); }

**class** C2 **extends** C1 {   
 **public** C2() { System.out.print(“2);   
 **super**(); }  
   
 **class** C3 **extends** C2 {  
 **public** C3() { System.out.print(“3”);  
 **super**(); }

**class** App {  
 **public static void** main(String[] args) {  
 **new** C3(); }  
 }

1. Error de compilación (explicar por qué)
2. 123
3. 321
4. Error en tiempo de ejecución (explicar por qué)
5. Ninguna de las anteriores (¿acadicealgoperonollegoaleer?)
6. Se dispone de las siguientes líneas de XXXXX donde se listan triángulos, rectángulos, cuadrados y círculos con sus atributos geométricos característicos.  
     
   triangulo(b1, h1).  
   triangulo(b2, h2).  
   cuadrado(l1).  
   cuadrado(l2).  
   rectangulo(b3,h3).  
   rectangulo(b4,h4).  
   circulo(r1).  
   circulo(r2).  
     
   Escribir las reglas prolog necesarias para encontrar la figura de área máxima. Ejemplificar la consulta.
7. Dada la siguiente secuencia de entrada:  
   25 7 54 38 2 1 50  
     
   a) Muestre como iría construyendo un montículo de máximo paso a paso, considerando inserciones sucesivas de cada elementos de la secuencia.  
     
   b) Muestre paso a paso los diferentes estados del montículo luego de eliminar dos veces consecutivas el máximo elemento.
8. Para cada uno de los siguientes fragmentos de código de un análisis en notación O del tiempo de ejecución.  
     
   a) **for** ( **int** i = 0; i < a; i -= 2) //parece que dice -= 2, pero no tiene mucho sentido que sea así  
     
   b)   
   **for** ( **int** i = 0; i < a; i++) {

**for** ( **int** j = a; j <= ……

**for** ( **int** k = 0; k < …..  
sum--;

Resolución

1. A) Es correcta, ya que es la creación de un nuevo objeto del tipo Guerrero, que se le es asignado a una referencia del tipo Guerrero. No hay ningún tipo de conflicto en esta asignación.

B) Esta asignación presenta un conflicto de tipos. No se le puede asignar a la referencia legolas, del tipo Guerrero, un objeto que sea de la clase Arquero, ya que son dos clases diferentes.   
  
C) Sucede lo mismo que en el anterior punto, solo que con clases diferentes.  
  
D) Esta asignación no presenta problemas. Si bien Recargable es una interfaz, y las interfaces no pueden ser instanciadas, acá no estamos instanciando una interfaz, sino que, estamos instanciando un objeto del tipo Catapulta. Y, como catapulta implementa la interfaz Recargable, la asignación puede hacerse sin problemas debido al polimorfismo.   
  
E) Esta asignación tampoco presenta problemas. Debido a la herencia, un arquero ES un personaje. Por lo tanto, no surgirán problemas al asignarle un objeto del tipo arquero a una referencia del tipo personaje.

1. La respuesta correcta es A, error de compilación. Esto es debido a que el super(); debe ser la primera llamada/instrucción del constructor. Esto es para asegurar que los atributos de la clase padre estén correctamente inicializados en caso de que necesitemos utilizarlos a través de un método de la clase hijo.   
     
   Acordarse de que, el compilador Java siempre inserta el super(); implícitamente, sin necesidad de que lo escribamos.
2. Ver: <https://github.com/AgustinCocciardi/Ejercicios_Prolog/blob/master/area_de_figuraJP.pl>
3. A) Los pasos para insertar la secuencia de números dada en un montículo máximo, utilizando una implementación del montículo con un array, del cual ignoramos la posición 0 del mismo, son los siguientes:

Secuencia: 25 7 54 38 2 1 50  
  
**cantelem:** variable que lleva cuenta de la cantidad de elementos del array. La utilizamos como índice indicador de posición. Se inicializa en 0.

Cómo obtener los padres e hijos a través del índice:

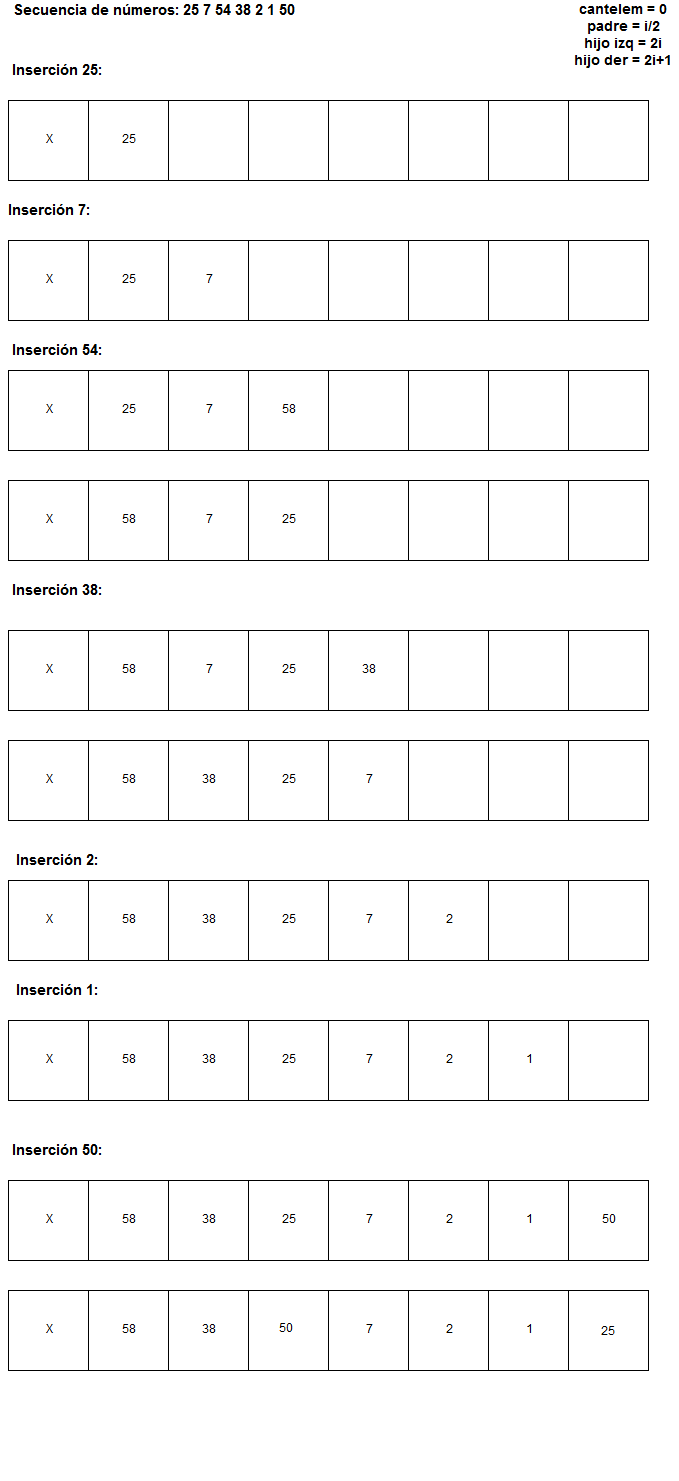
**padre = i/2  
hijo izq = 2i  
hijo der = 2i+1**

Inserción del número 25:   
  
-Insertamos el número en la posición cantElem+1.   
-Aumentamos la cantidad de elementos en 1.  
-Declaramos a i = cantElem.

-Mientras que el padre sea menor que 25, y mientras que i sea mayor que 1, intercambiamos de posición al padre con el hijo, y asignamos i = i/2: Como i es igual a 1, esto nos indica que 25 es el primer elemento, por lo tanto, no se producen ningún tipo de intercambios.   
  
Inserción del número 7:

-Insertamos el número en la posición cantElem+1.  
-Aumentamos la cantidad de elementos en 1.  
-Declaramos a i = cantElem.  
-Mientras que el padre sea menor a 7, y mientras que i sea mayor a 1, intercambiamos de posición al padre con el hijo, y asignamos i = i/2: Como 7 es menor que 25, no se produce ningún intercambio.  
  
Inserción del número 54:   
  
-Insertamos el número en la posición cantElem+1.  
-Aumentamos la cantidad de elementos en 1.  
-Declaramos a i = cantElem.  
-Mientras que el padre sea menor a 7, y mientras que i sea mayor a 1, intercambiamos de posición al padre con el hijo, y asignamos i = i/2: Como el padre es menor que el hijo, necesitamos ajustar el montículo. Intercambiamos al padre con el hijo.  
  
Y así sucesivamente con los demás números. Acá un diagrama de cómo sería la inserción de todos los demás números en el montículo.

En el gráfico me confundí y puse 58 en vez de 54. No afecta en nada.



B) Para quitar el elemento del montículo, ejecutamos los siguientes pasos:  
  
-Seteamos el valor de retorno como el primer elemento del mismo, es decir, su raíz.   
-Sobreescribimos la raíz con el último elemento del montículo.

-Llamos a adjustar\_monticulo() teniendo en cuenta que ahora el tamaño del montículo decreció una unidad.  
-Retornamos el valor.   
  
adjustar\_monticulo() puede hacerse de forma recursiva como iterativa. Sin embargo, creo que la forma recursiva es la que les gustaría más a los profes, por lo que dejo esa.

Adjustar montículo se realiza de la forma siguiente:

-Si el nodo que estamos evaluando es una hoja, no hacemos nada, retornamos.  
Para chequear si el nodo que estamos evaluando es una hoja, preguntamos si su posición es mayor o igual a cantElem/2 y si es menor o igual a cantElem.  
  
-Si el nodo que estamos evaluando es menor que su hijo izquierdo, o su hijo derecho: intercambiamos el nodo con el mayor de los hijos. Luego llamamos de nuevo a adjustar\_monticulo, con la posición del intercambio.

Abajo una imagen que muestra el estado del montículo paso a paso.

