ALUMNO: LEON PACHECO CRISTIAN IRVING GRUPO:3SB HORARIO: 8:00 AM – 9:00 AM

EDD U1 Eval. 1ª op.

Lectura del documento "04 01 3.3 Designing Data Types.pdf" para resolver el siguiente cuestionario:

1. ¿Qué mecanismos de programación se sugieren para diseñar tipos de datos?
2. ¿Qué se consigue si aplicamos correctamente estos mecanismos?
3. Describe qué es el encapsulamiento.
4. ¿Como se consigue que un TDA construya objetos encapsulados?
5. Describe qué es la inmutabilidad.
6. ¿Como se consigue que un TDA construya objetos inmutables?
7. ¿Para qué sirve la referencia this?
8. En Java se puede implementar la herencia de dos maneras: por subtipos o por subclases.
   a) ¿En qué consiste la herencia por subtipos?
   b) ¿Cómo se programa?
9. ¿Cómo se define una interface?
10. ¿Cómo se implementa?
11. ¿Cómo se usa?
12. ¿Para que se usan las interface Comparable, Iterator e Iterable de Java?
13. ¿En qué consiste la herencia por subclases?
14. ¿Porqué se debería evitar la herencia por subclases al diseñar un TDA?
15. ¿Cuál es la utilidad de redefinir el método toString() heredado de Object?
16. ¿Para qué se usa el método equals() heredado también de Object?
17. Describe qué son y cuáles son los "wrapper types" de Java.
18. ¿Para qué se usan?
19. ¿Qué es el Autoboxing?
20. ¿Y Qué es el Unboxing?
21. Describe qué son las excepciones.
22. Describe qué son las aserciones.

Notas importantes:

a) En lo posible escribe "tus" propias respuestas; es decir, no copies-pegues lo que viene en el documento.

b) Escribe tus respuestas en un documento pdf sin portada, usa el área de encabezado para anotar tus datos personales y en el pie de página agrega la paginación.

c) Si consultas "algo" en internet, anota la liga del sitio web visitado como referencia.

d) Procura no equivocarte, estás aprendiendo, pero no esperes a la 2ª oportunidad.

ALUMNO: LEON PACHECO CRISTIAN IRVING GRUPO:3SB HORARIO: 8:00 AM – 9:00 AM

RESPUESTAS:

1.- Los mecanismos que son utilizados para diseñar nuestros tipos de datos son el encapsulamiento la inmutabilidad y la herencia.

2.- Si usamos de manera correcta estos mecanismos que aplicamos para crear nuestros diferentes tipos de datos con base a nuestra programación modular podemos tener códigos mas claros y correctos.

3.- Mediante el encapsulamiento podemos tener un correcto proceso de separación de clientes de las implementaciones que se realizan ocultando toda esta información es decir ocultamos cada uno de los atributos de nuestro objeto para que estos solo puedan ser cambiados con operaciones específicas.

4.- La manera mas correcta es declarando nuestras variables de instancia como privadas.

5.- La inmutabilidad es cuando no es posible modificar las instancias y valores que tienen variables de nuestros objetos que estemos utilizando, un ejemplo muy claro de esto y que alguna vez hemos hecho es la clase String.

6.- Podemos construir objetos de tipos imputable con base al TDA por medio de nuestro constructor ya que este nos pide parámetros que se asignan a la variable de instancia las cuales son private y final, de esta manera al crear nuestro objeto lo hacemos inmutable.

7.- Esta nos ayuda a hacer referencia a las variables de instancia del objeto actual que están siendo llamadas, además que nos permite modificar nuestro primer parámetro de un modo extensión.

8.-

a) Esta herencia se da por medio de una relación que existe entre las clases de esta manera podemos tener los métodos en ambas clases que comparten parámetros en común.

b) Lo podemos programar de una manera sencilla:

public interface<nombre_de_la_interface_que_manejamos>{

<métodos abstractos> }

9.- Nuestra interface para poder ser definida es necesario que contenga métodos abstractos y esta se debe guardar con el mismo nombre de la interface con nuestra extensión .java.

10.- La manera de implementar la clase es poniendo la palabra “implements” seguido del nombre de nuestra interface. Implementando cada uno de nuestros métodos en la interface.

11.- Al ser nuestra interface un tipo de referencia podemos declarar cualquier tipo de variable para que sea el nombre de nuestra interfaz. De esta manera cualquier objeto que asigne a esa variable debe ser una instancia de una clase que implemente la interfaz. Nuestro lenguaje java nos permite al momento de invocar a nuestros métodos por medio de nuestras varibales de interfaz java nos ayuda a encontrar el método correcto que se requiera utilizar

12.-

a) La interface comparable es usada mayormente para comparar nuestros objetos del mismos tipo, ya sea como el orden alfabetico en cada una de nuestras cadenas o el orden ascendente de los números enteros, para eso se usa nuestra comparación.

b) La interface intertor e interable nos ayuda a los usuarios o clientes a iterar cada uno de los elementos sin depender de la representación.

13.- Consiste en heredar variables de instancia atributos y métodos de instancia o permitir la reutilización de nuestros códigos.

14.-Debido a que existen algunas subclases remanentes integradas en java, al programar en nuestro lenguaje java se anulan uno o mas de estos métodos.

15.-La utilidad es que la convención es la base para que java convierta automáticamente el operando el operador de concentración de cadenas en una cadena siempre que el otro sea una cadena.

16.-Esto nos ayuda comprobar si dos objetos son iguales.

17.- Los métodos toString(), hashCode() y equals son identificadores que solo funcionan con tipos de referencia y no con tipos primitivos

18.- Son utilizados principalmente para representar valores de tipos primitivos como objetos.

19.- Es el proceso que se lleva acabo cuando un tipo primitivo es automáticamente encapsulado en su envoltura de tipo equivalente cuando se necesita un objeto de ese tipo.

20.- Es el proceso por el cual el valor de un objeto encapsulado se extrae automáticamente es decir se desencapsula de una envoltura de tipo cuando se necesita su valor.

21.- Una excepción es básicamente un error o un evento disruptivo que ocurre mientras se ejecuta un programa, a menudo para señalar un error. Existen excepciones por varios factores que se muestran en nuestro código ya como una mala redacción, o una excepción muy básica como dividir entre de 0.

22.- Una aserción es una expresión booleana que afirma que es verdadera en algún momento durante la ejecución de un programa. Si la expresión es falsa, el programa generará un AssertionError , el cual finalizara nuestro programa y nos informara del error que se ha cometido.