Universidad Tecnológica Nacional
TSB – Consignas para el Examen Final
(Vigentes desde Diciembre de 2013 en adelante)

Este documento de consignas de examen final entra en vigencia a partir de los turnos de exámenes de ***Noviembre / Diciembre de 2013*** para la asignatura TSB – Electiva Tercer Nivel. Siguen vigentes los enunciados y consignas propuestos para el cursado 2012, pero recordamos que cualquier consigna de examen final *anterior a 2012* (que no esté renovada en este documento) DEJÓ DE TENER VIGENCIA en los turnos de Febrero / Marzo de 2013. Vale decir: los alumnos que cursaron en 2012 podrán seguir usando los enunciados que se les sugirieron oportunamente, pero ahora también pueden elegir cualquiera de los enunciados y consignas que se sugieren en este.

Para rendir el examen final de la asignatura, los alumnos de TSB deberán presentar un trabajo final que podrán realizar en grupos en forma análoga a lo realizado durante el cursado. No hay límite de tiempo para la entrega del trabajo (salvo que cambien los enunciados de las consignas en algún período posterior), pero está claro que mientras más pronto se entregue, más valor tendrá para cada uno el esfuerzo.

La cátedra sugiere algunos enunciados exigentes, y los alumnos deberán seleccionar el que prefieran. En todos los casos, se espera que se apliquen en la mayor medida los temas y conceptos vistos a lo largo de la materia, pero también debe quedar claro que tendrán que hacer un fuerte trabajo de investigación de nuevos temas, no necesariamente vistos en clase.

Obviamente, los alumnos que cursaron la materia en 2013 no tienen opción: solo pueden usar los enunciados previstos en este documento, o sugerir alguno adicional a sus profesores. Será válido que un grupo proponga algún trabajo a desarrollar, para lo cual deberán dirigirse con la debida antelación a cualquiera de los profesores de la cátedra, y presentar su idea para ser aprobada ANTES de ponerse a trabajar.

Para presentar el trabajo final, simplemente deben inscribirse para rendir en forma normal, y el día del examen traer su trabajo para ser expuesto frente al profesor que esté a cargo del examen. Se deja expresamente claro que podrán trabajar en grupos, pero que el día del examen final cada alumno será llamado por separado y deberá explicar en forma individual lo que se haya implementado, pudiendo los profesores hacer cualquier tipo de pregunta pertinente. La experiencia indica que el día del examen, lo más conveniente es que cada grupo traiga a la Facultad su propia computadora con el sistema presentado listo para funcionar.

Por el momento, estos son los temas propuestos por la Cátedra a modo de consignas para el examen final (pueden aparecer otros en los próximos días):

---

a. Desarrollar una implementación propia de la clase *Hashtable*, en base a las siguientes líneas:

- La implementación debe basarse en la técnica de *direccionamiento abierto* para el tratamiendo de las colisiones: si dos objetos son mapeados al mismo lugar de la tabla, debe explorarse en la misma tabla hacia abajo la existencia de otro lugar libre (y NO usar listas de desborde como se hizo en los modelos analizados en clase).
- La exploración de la tabla para buscar el próximo casillero libre, DEBE hacerse con *exploración cuadrática* (Y NO con exploración lineal). Por lo tanto, la implementación debe incluir todos los elementos que garanticen que la búsqueda tenga éxito.
- En la implementación *debe* usarse el mecanismo de generics para control de homogeneidad de contenidos.
- La clase *Hashtable* que se proponga, debe implementar la interfaz *java.util.Map* que Java ya provee para este tipo de colecciones. Sugerimos tomar como modelo la documentación

javadoc de la clase Hashtable y de la interfaz Map para que sirvan como "lista de control" de los métodos y funcionalidades esperadas.

- La clase desarrollada por los alumnos debe ser tan "fiel" como sea posible a la clase *java.util.Hashtable*, de forma que eventualmente puedan reemplazarse entre ellas sin provocar error de compilación (obviamente, haciendo el ajuste en el import que corresponda). Por lo tanto, insistimos: use como modelo la documentación javadoc de *java.util.Hashtable* y *java.util.Map* para no olvidar detalle alguno.
- El trabajo debe hacerse en base a ideas y conceptos vistos en clase o investigados por el alumno en diversas fuentes y referencias bibliográficas. Hacemos notar que el JDK provee los archivos fuente de las clases nativas de Java, pero para esta implementación los alumnos NO DEBEN abrir el fuente de la clase *java.util.Hashtable* para tomar ideas desde allí, sino resolver los detalles de la implementación con criterios y conocimientos propios o adquiridos en fuentes externas. Es decir, se debe trabajar en condiciones conocidas como "*clean room*": los desarrolladores no deben haber tenido acceso activo al código fuente de ninguna *implementación profesional* previa conocida para la clase pedida (ni de Sun, ni de Oracle, ni de un tercero, incluyendo fuentes open source) ni tener acceso a dicho código fuente durante el desarrollo que se proponga. Obviamente, son válidas las fuentes basadas en libros de texto, notas de clase, papers y presentaciones en revistas o congresos, etc. El día del examen, los profesores se tomarán su tiempo para controlar que este punto efectivamente se respete.
- Tiempo estimado de desarrollo, al leal saber y entender de los profesores, considerando un esquema de entre tres y cuatro horas de trabajo por día e incluyendo tareas de investigación, estudio, diseño, implementación y prueba: 30 días. A modo de referencia comparativa, hacemos notar que la implementación de una tabla hash basada en direccionamiento abierto ya fue pedida a los alumnos en otras oportunidades en años anteriores (aunque con menos exigencias) a modo de trabajo práctico, para ser entregado en no más de tres semanas de plazo.
- Bibliografía básica: la siguiente es una lista de posibles fuentes de consulta para la implementación de la clase *Hashtable*.

  1.) Documentación javadoc de la clase Hashtable. Puede verse en el siguiente url: http://docs.oracle.com/javase/1.5.0/docs/api/java/util/Hashtable.html.

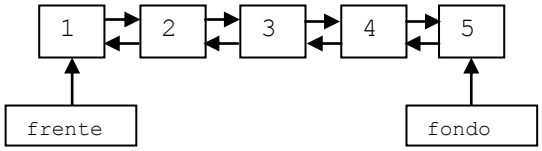
  2.) Langsam, Y., Augenstein, M., y Tenenbaum, A. (1997). "Estructura de Datos con C y C++ (2da. Edición)" - Páginas 468 y siguientes. México: Prentice Hall. ISBN: 968-880-798-2 [disponible en biblioteca central de la UTN Córdoba]

  3.) Weiss, M. A. (2000). "Estructuras de Datos en Java – Compatible con Java 2" – Capítulo 19. Madrid: Addison Wesley. ISBN: 84-7829-035-4 [disponible en biblioteca central de la UTN Córdoba]

---

b. Sabemos que el lenguaje Java provee tipos primitivos para representar números enteros de hasta 8 bytes de precisión (con el tipo *long*) y que además existen las clases de envoltorio para representar a los primitivos como objetos (la clase *Long* permite representar los números de tipo *long* como objetos). Sin embargo, si requiere trabajar con números enteros muy grandes, que excedan la precisión de 8 bytes, el tipo *long* no será suficiente. Una opción es trabajar con tipos de coma flotante, y si eso tampoco alcanza Java provee la clase *java.math.BigInteger* que permite representar y manipular números enteros de precisión arbitraria.

En esta consigna, se solicita que el estudiante diseñe y desarrolle una clase propia llamada también *BigInteger* para representar números enteros muy grandes. Una forma de hacerlo, consiste en representar a cada número entero con una lista doblemente vinculada, de forma que cada nodo contenga a uno de los dígitos del número representado. Así, por ejemplo, el número 12345 podría ser representado por la siguiente lista:



La implementación debe basarse en las siguientes líneas:

- Los alumnos tienen libertad de elegir la estructura de soporte para la implementación de la idea. En ocasión del Parcial Unico 2013 (turno mañana – curso 3k1) se pidió hacer una implementación muy acotada basada en el uso de una lista en la que cada nodo contuviese un dígito del número representado, pero esa idea (si bien correcta) no es obligatoria. Los alumnos podrán explorar otras ideas (u otras maneras de implementar la misma que se sugirió en el parcial).
- La clase *BigInteger* que se proponga, debe derivar de la clase *java.lang.Number* que Java ya provee para sus propias clases que representan números de precisión arbitraria. Sugerimos tomar como modelo la documentación javadoc de la clase *java.math.BigInteger* y de la clase *java.lang.Number* para que sirvan como "lista de control" de los métodos y funcionalidades esperadas.
- La clase desarrollada por los alumnos debe ser tan "fiel" como sea posible a la clase *java.math.BigInteger*, de forma que eventualmente puedan reemplazarse entre ellas sin provocar error de compilación (obviamente, haciendo el ajuste en el *import* que corresponda). Por lo tanto, insistimos: use como modelo la documentación javadoc de *java.math.Hashtable* y *java.lang.Number* para no olvidar detalle alguno.
- El trabajo debe hacerse en base a ideas y conceptos vistos en clase o investigados por el alumno en diversas fuentes y referencias bibliográficas. Hacemos notar que el JDK provee los archivos fuente de las clases nativas de Java, pero para esta implementación los alumnos NO DEBEN abrir el fuente de la clase *java.math.BigInteger* para tomar ideas desde allí, sino resolver los detalles de la implementación con criterios y conocimientos propios o adquiridos en fuentes externas. Es decir, se debe trabajar en condiciones conocidas como "*clean room*": los desarrolladores no deben haber tenido acceso activo al código fuente de ninguna *implementación profesional* previa conocida para la clase pedida (ni de Sun, ni de Oracle, ni de un tercero, incluyendo fuentes open source) ni tener acceso a dicho código fuente durante el desarrollo que se proponga. Obviamente, son válidas las fuentes basadas en libros de texto, notas de clase, papers y presentaciones en revistas o congresos, etc. El día del examen, los profesores se tomarán su tiempo para controlar que este punto efectivamente se respete.
- Tiempo estimado de desarrollo, al leal saber y entender de los profesores, considerando un esquema de entre tres y cuatro horas de trabajo por día e incluyendo tareas de investigación, estudio, diseño, implementación y prueba: 45 días.
- Bibliografía básica: la siguiente es una lista de posibles fuentes de consulta para la implementación de la clase *BigInteger*.

4.) Documentación javadoc de la clase BigInteger. Puede verse en el siguiente url: http://docs.oracle.com/javase/1.4.2/docs/api/java/math/BigInteger.html.

5.) Números de Precisión Arbitraria: en Wikipedia (versión en inglés) se expone un resumen bastante completo: http://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic.

6.) Langsam, Y., Augenstein, M., y Tenenbaum, A. (1997). "Estructura de Datos con C y C++ (2da. Edición)" - Páginas 235 y siguientes. México: Prentice Hall. ISBN: 968-880-798-2 [disponible en biblioteca central de la UTN Córdoba]

7.) Sedgewick, Robert (1995). "Algoritmos en C++" - Capítulo 36. Reading: Addison Wesley – Díaz de Santos. ISBN: 978-0-201-62574-5 [disponible en biblioteca central de la UTN Córdoba]