# Language Enhancements

En un Try Con recursos cuando se genera una excpetion, primero se cierran los recursos y después de capturan la excepciones

En los case de un swith puede haber variables solo si son final.

Al declarar una variable primero va <> y después el nombre de la variable

Un float, o un doublé, no se puede escribir en binario con letra al final

Un recurso definido en el try no puede ser redefinido

statements are true about the diamond operator:

* It reduces code clutter and improves readability.
* It has no impact on the run time execution of the code.

that String comparison is case sensitive in a switch statement.

Remember the following points about a multi-catch:

* Exception classes are separated by |
* There is only one variable declared at the end.
* The classes in a multi catch must not be related by inheritance. i.e. one exception class must not be a subclass/superclass of any other exception in the list.
* In the catch block you can only access those properties of the exception which are present in the most specific super class of all the exceptions in the list. For example, if you have SQLException and IOException in the list, you cannot access any property that is defined in SQLException or IOException. You can only access those properties that defined in Exception class because Exception class is the most specific class that is a super class of both SQLException and IOException.

You need to know the following points regarding try-with-resources statement for the exam:

1. The resource class must implement java.lang.AutoCloseable interface. Many standard JDK classes such as BufferedReader, BufferedWriter) implement java.io.Closeable interface, which extends java.lang.AutoCloseable.

2. Resources are closed at the end of the try block and before any catch or finally block.

3. Resources are not even accessible in the catch or finally block. For example:

   try(Device d = new Device()) {    d.read();     }

finally{

      d.close(); //This will not compile because d is not accessible here.

     }

4. Resources are closed in the reverse order of their creation.

5. Resources are closed even if the code in the try block throws an exception.

6. java.lang.AutoCloseable's close() throws Exception but java.io.Closeable's close() throws IOException.

7. If code in try block throws exception and an exception also thrown while closing is resource, the exception thrown while closing the resource is suppressed. The caller gets the exception thrown in the try block.

Los recursos en un try con recursos deben ser creados dentro de los parentesis

# Design Patterns

The Singleton patterns enforces that only one object of the class is ever created. To achieve this, you need to do 3 things:

1. Make the constructor of the class private so that no one can instantiate it except this class itself.
2. Add a private static variable of the same class to the class and instantiate it.
3. Add a public static method (usually named getInstance()), that returns the class member created in step 2.

La delegación de métodos es un aspecto importante en el patrón Composite.

La herencia es opuesta a Composition

La herencia y composition son técnicas para agregar características a clases en java

Composition es muy utilizado cuando la clase ya extends de otra clase.

Composition and Inheritance both are techniques for code reuse.

Un Factory nos sirve para separar la dependencia de instanciar un objeto determinado. Así podemos eliminar el acoplamiento en la clase.

Cuando se crea una instancia sin la palabra reservada new, se pueden pensar en el patrón Factory o en el patrón Singleton. La diferencia entre estos es que un singleton devuelve la misma instancia.

Beneficios de DAO

* Persistence logic is factored out of the domain classes.
* It reduces code complexity in business objects.

Sobre DAO

* Persistence logic is factored out of the domain classes.
* It reduces code complexity in business objects.

Data Access Object Pattern is best suited when...

* the data clients have to be shielded from the underlying database.
* the persistent data storage may have to be changed later.

Data Access Object pattern abstracts the data storage and provides a uniform way to access the data. This is very useful in situations when the data storage may change like from file-based to LDAP to relational database and so forth.

El patrón DAO se utiliza para separar la lógica de negocio con el acceso a la base de datos

El patrón Factory esta basado en el principio "program to an interface"

No te dejes engañar, DriverManager.getConnection utiliza Factory.

Utiliza Factory cuando estas creando una aplicación la cual se conecta a diversas fuentes, esto es para cuando no conoces que dataobjecacces debe ser instanciado

Utiliza una clase abstracta cuando :

* You need to have a root class for a hierarchy of related classes.
* You want to pass different implementations for the same abstraction in method calls.

Utiliza una interface cuando

* The implementation details are not known at the time.
* You are only modeling the behavior of an abstract concept.

Para aplicar el DAO patrón se deben hacer dos cosas.

* Move find, save, remove, and update methods to another class.
* Create an interface that contains the signatures of find, save, remove, and update methods.

DAO elimina la exposición de sentencias SQL de la aplicación

Verdades sobre Factory:

* It reduces inherent tight coupling associated with instance creation.
* It eliminate direct constructor calls and allows the clients to use a regular method instead.
* It is a creational pattern.
* It reduces compile time dependency on actual implementation classes.
* It prevents exposure of instantiation logic of a class to other classes.

# Database applications with JDBC

Sobre CallableStatement

* A CallableStatement is the only way for a Java program to execute stored procedures in the database if the procedure has in and out parameters.
* One advantage of CallableStatement is that it allows IN/OUT parameters.
* A CallableStatement is easier to build and call from JDBC code than a PreparedStatement.

Sobre RowSet interfaces.

* FilteredRowSet provides a connected view of the database.

Sobre fork/join framework

Incrementa e throughput para el CPU en tareas intensivas

Una mejora es que es paralelizable

Un ResultSet solo no es actualizable. Y es de solo lectura generalmente

El estándar SQL del JDBC es el SQL92

Sobre JDBC

* It is used to interact with a relational database using SQL based queries.
* It allows queries to be built dynamically at run time.

Un JDBC debe provver implementación de las clases

* DriverManager
* Connection
* Statement

JDBC API provides multiple sub-interfaces of RowSet interface.

PreparedStatement

* PreparedStatement offers better performance when the same query is to be run multiple times with different parameter values.
* PreparedStatement allows several additional SQL types such as BLOB and CLOB.

PreparedStatement offers protection against SQL injection attacks.

Tenga en cuenta que con JDBC 4.1 en adelante, no hay necesidad de cargar la clase del controlador de forma explícita utilizando Class.forName. El archivo jar que contiene la clase de controlador contiene la información sobre el nombre de clase del controlador y una vez que el frasco se coloca en la ruta de clases, la clase del controlador dado se carga automáticamente.

Para usar un JDBC se debe colocar el driver en el classpath y llamar al método DriverManager.getConnection()

El método roolback asegura que se restaurara hasta el punto marcado si no se hizo commit antes.

Si se cierra un Statement que devuelve un resulset este también será cerrado.

El método getColumName de ResultSet incia en 1, cualquier otro caso genera una exception

Predicate interface sirve para crear un filtro en un rowset, se implementa el método evaluate(); este método filtrara a los objetos cuyo método evaluate retorne true;

El driver JDBC NO necesita ser cargado explícitamente en ninguna parte de la aplicación por el método DriverManager.classForName();

El commit esta activado por defecto en todas las connecciones que se crean.

Un código catch vacío no es una buena practica.

Un CallableStatement se usa para ejecutar un procedimiento almacenado en una base de datos.

java.sql.CallableStatement extends PreparedStatement +

# Concurrency

Un WatchKey se convierte en invalido hasta que es cancelado o cerrado en WatchService

Fork/join

* The worker threads in the ForkJoinPool extend java.lang.Thread and are created by a factory.
* One worker thread may steal work from another worker thread.

CopyOnWriteArrayList es un array list que te permite add y remove seguros

CachedRowSet si necesitas enviar datos por la red.

El numero de instancias del executor, depende de como fue instanciado el executor.

Cuando se necesita devolver un valor se utiliza RecursiveTask y cuando no es necesario devolver el valor se usa RecursiveAction

**Tareas que no retornan nada**

Executor.- solo acepta Runnables

Runnable.- no retorna nada

**Tareas que retornan valor**

Executors.- clase con utilidades para construir diversos

Callable.- devuelven un valor

Future.- se utiliza para recuperar un Callable.

ExecutorService.- puede tomar objetos Callables

**Metodos de AtomicVariables**

addAndGet();

incrementAndGet();

El método compareAndSet(firts,second) verifica que la variable tenga el valor del primer parámetro y después coloca el valor del segundo parámetro.

El método call de Callable lanza una Exception, run no.

En un ReentrantLock exiten los métodos lock() y tryLock() los cuales retornan void y boolean respectivamente.

Executors. newFixedThreadPool(4); devuelve un ExecutorService

Un callable no puede ser pasado en el constructor de Thread

ForkJoinPoll es el mas eficiente pata tareas recursivas

**RecursiveAction**

* Process
* Split
* invoke

**Fork/join**

El orden y de join y compute es critico

Es usado en algoritmos computacionales intensivos. Utiliza la recursividad.

El numero de hilos creados depende de como el fork/join es creado. por defecto usa el numero de procesadores disponibles.

El fork/join crea subprocesos que pueden generar nuevas tareas independientes. A esto se le conoce como robo de trabajo.

ForkJoinPool es el mas eficiente para ejecutar tareas recursivas.

Cuando dos trates intentan obtener datos de un colección de una misma instancia pueden ocurir:

Que se genere un IndexOutOfBoundsException.

Que se obtengan dos intancias distintas,

Que las dos tareas obtengan el mismo objeto

**RecursiveTask**

El método compute debe retornar un valor

A la tarea1 se le aplica el método compute y a la tarea2 el método join. Se debe retornar la suma de estos dos en el else

concurrent collection interfaces are provided by JDK 7

* BlockingQueue
* ConcurrentMap
* ConcurrentNavigableMap

Sentencias validas sobre fork/join

* It is well suited for computation intensive tasks that can be broken into smaller pieces recursively.
* A ForkJoinPool differs from other kinds of ExecutorService mainly by virtue of employing work-stealing.

# Localization

Un Locale se construlle con un lenguaje y un country.

Las claves en un archivo propierties siempre se gurdan como string

Cuando un programa carga un resource blunde el archivo propierties siempre debe estar en el CLASSPATH y el resurce debe ir indicado opr un \_language\_COUNTRY

Un locale provee mecanismos para formatear fechas y monedas.

Tres formas validas para crear un Locale

* Locale myLocale = Locale.getDefault();
* Locale myLocale = Locale.US;
* Locale myLocale = new Locale("ru", "RU");

Para imprimir el nombre completo de un mes se usa MMMM y yy para el Año

Para un SimpleDateFormat se utiliza zzzz para un formato full o completo

Un objeto Locale Representa: a specific geographical, political, or cultural region.

# NIO 2

BasicFileAttributeView

|  |  |
| --- | --- |
| * "lastModifiedTime" | * FileTime |
| * "lastAccessTime" | * FileTime |
| * "creationTime" | * FileTime |
| * "size" | * Long |
| * "isRegularFile" | * Boolean |
| * "isDirectory" | * Boolean |
| * "isSymbolicLink" | * Boolean |
| * "isOther" | * Boolean |
| * "fileKey" | * Object |
| DosFileAttributeView   |  |  | | --- | --- | | * readonly | * Boolean | | * hidden | * Boolean | | * system | * Boolean | | * archive | * Boolean | |  |

El \*\* indica cualquier carácter y cruza las fronteras del directorio.

PosixFileAttributeView apoya en la manipulación de archivos de un propiedario.

En un WathcService hay tres métodos

* Poll()
* Poll(long time, timeUnite)
* Take()

Para recuperar los atributoss de un archivo en Windows se usa la clase DosFileAttributes

El método relativize solo recibe como argumentos directorios, otro Path generara una IlegalArgumentException

Remember the following 4 points about Path.getName() method :  
  
1. Indices for path names start from 0.  
2. Root (i.e. c:\) is not included in path names.  
3. \ is NOT a part of a path name.  
4. If you pass a negative index or a value greater than or equal to the number of elements, or this path has zero name elements, java.lang.IllegalArgumentException is thrown. It DOES NOT return null.

Files.isSameFile(p1, p2) verifica si los path son el mismo archivo

Sobre WathService

You do not need to specify the OVERFLOW event while registering a watchable with a WatchService  to receive this event.

The counts for ENTRY\_CREATE and ENTRY\_DELETE are always 1.

Para ocultar un archivo utiliza la clase Files

java.nio.file.PathMatcher contiene un único método boolean matches(Path path)

FileVisitResult

* CONTINUE Continue.
* SKIP\_SIBLINGS Continue without visiting the siblings of this file or directory.
* SKIP\_SUBTREE Continue without visiting the entries in this directory.
* TERMINATE Terminate.

WatchService

* ENTRY\_CREATE
* ENTRY\_DELETE
* ENTRY\_MODIFY
* OVERFLOW

WhatchKey

* Signaled
* Ready
* Invalid

StandardOpenOption. Combinaciones validas.

* StandardOpenOption.WRITE,StandardOpenOption.DSYNC
* StandardOpenOption.APPEND, StandardOpenOption.SYNC

StandardOpenOption

* APPEND If the file is opened for
* WRITE access then bytes will be written to the end of the file rather than the beginning.
* CREATE Create a new file if it does not exist.
* CREATE\_NEW Create a new file, failing if the file already exists.
* DELETE\_ON\_CLOSE Delete on close.
* DSYNC Requires that every update to the file's content be written synchronously to the underlying storage device.
* READ Open for read access.
* SPARSE Sparse file.
* SYNC Requires that every update to the file's content or metadata be written synchronously to the underlying storage device.
* TRUNCATE\_EXISTING If the file already exists and it is opened for WRITE access, then its length is truncated to 0.
* WRITE Open for write access.

El método getRootDirectories(); retorna un iterable de paths, no un iterator

La sintaxis para ocultar un archivo con el método setAtributesFile es (path,view-name:attribute-name,true,standaroptions)

Sobre relativaze:

Path p1 = Paths.get("home\\clarence");// unir dos paths el que invoca al metodo debe tener el root

System.out.println(p1.resolve("clarence\\js2"));

Path p2 = Paths.get("home/clarence");// si no tiene el root el path final queda con la segunda ruta

System.out.println(p2.resolve("/foo"));

Path p3 = Paths.get("/home/clarence");// si los dos tiene root el path final queda con la segunda ruta

System.out.println(p3.resolve("/foo"));

Path p4 = Paths.get("home/clarence"); // si ninguno tiene el root el segundo se agrega al primero

System.out.println(p4.resolve("foo"));

Path p1 = Paths.get("c:\\temp\\test.txt");  
   Path p2 = Paths.get("c:\\temp\\report.pdf");

//agrega el final al primer path