Почему в C++ нет finally?

- C++ использует подход RAII[1,2]: https://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization (Получение ресурса есть инициализация)
- Все объекты, созданные на стеке, будут гарантировано уничтожены, а их ресурсы могут освобождаться в деструкторах
- Плюсы: более простой код использования ресурсов, переиспользование кода освобождения ресурсов (считается, что мест получения ресурсов больше, чем самих ресурсов)
- Ограничения: работает для ресурсов, создаваемых на стеке. Для динамической памяти нужны свои подходы (см. https://en.wikipedia.org/wiki/Smart_pointer)
- Вопрос: почему RAII нет в Java?

1) http://stackoverflow.com/questions/161177/does-c-support-finally-blocks-and-whats-this-raii-i-keep-hearing-about
2) http://www.stroustrup.com/bs_faq2.html#finally

Пример RAII[1]

```
// A class with implements RAII
class lock
{
    mutex &m_;

public:
    lock(mutex &m)
      : m_(m)
    {
        m.acquire();
    }
    ~lock()
    {
        m_.release();
    }
};
```

```
// A class which uses 'mutex' and 'lock' objects
class foo
{
    mutex mutex_; // mutex for locking 'foo' object
public:
    void bar()
    {
        lock scopeLock(mutex_); // lock object.

        foobar(); // an operation which may throw an exception

        // scopeLock will be destructed even if an exception
        // occurs, which will release the mutex and allow
        // other functions to lock the object and run.
    }
};
```

1) http://stackoverflow.com/questions/161177/does-c-support-finally-blocks-and-whats-this-raii-i-keep-hearing-about

Освобождение ресурсов

- Ресурсы: процессорное время (CPU), память, диск, сеть, блокировки
- Примеры: бесполезные циклы for + sleep/wait/poll (polling, spinning), буферы в памяти, открытые файловые дескрипторы, сетевые соединения, lock-объекты и т.д.
- Нельзя допускать утечек ресурсов (leaks)
- Желательно сводить к минимум время использования ресурса

Пример:

```
static String readLine() throws IOException {
    BufferedReader in = new BufferedReader(new FileReader("tmp/in.txt"));
    try {
        return in.readLine();
    } finally {
        in.close();
    }
}
```

https://docs.oracle.com/javase/tutorial/essential/exceptions/finally.html

try-with-resources

```
void copyStr() throws IOException {
    try (BufferedReader in = new BufferedReader(new FileReader("in.txt"));
         BufferedWriter out = new BufferedWriter(new FileWriter("out.txt"))) {
        out.write("Copy: " + in.readLine());
    }
}
```

- Синтаксис: try (<? extends AutoCloseable> res = <init> [; <? extends AutoCloseable> resN = <init>]*)
- AutoCloseable: void close() throws Exception
  - Реализации могут не объявлять генерируемые исключения, тогда и блок try-with не будет требовать обработать или пробросить их
- Throwable.getSuppressed – получение “подавляемых” исключений

https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Почему Exceptions?

Задача (псевдо-код):

```
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

https://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html

Решения по обработке ошибок

Old school

```
errorCodeType readFile {
   initialize errorCode = 0;

   open the file;
   if (theFileIsOpen) {
      determine the length of the file;
      if (gotTheFileLength) {
         allocate that much memory;
         if (gotEnoughMemory) {
            read the file into memory;
            if (readFailed) {
               errorCode = -1;
            }
         } else {
            errorCode = -2;
         }
      } else {
         errorCode = -3;
      }
      close the file;
      if (theFileDidntClose && errorCode == 0) {
         errorCode = -4;
      } else {
         errorCode = errorCode and -4;
      }
   } else {
      errorCode = -5;
   }
   return errorCode;
}
```

Exceptions

```
readFile {
   try {
      open the file;
      determine its size;
      allocate that much memory;
      read the file into memory;
      close the file;
   } catch (fileOpenFailed) {
      doSomething;
   } catch (sizeDeterminationFailed) {
      doSomething;
   } catch (memoryAllocationFailed) {
      doSomething;
   } catch (readFailed) {
      doSomething;
   } catch (fileCloseFailed) {
      doSomething;
   }
}
```

https://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html