Exception Handling in Threads

This lesson discusses exception handling in threads.

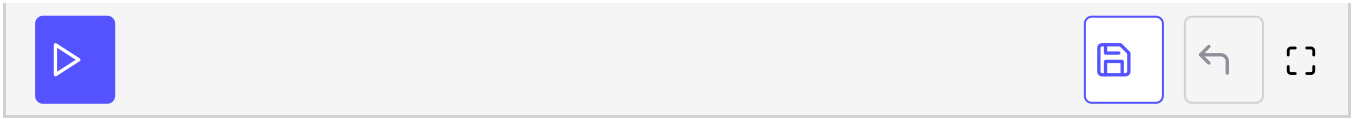
Exception Handling in Threads

Unhandled exceptions can occur in spawned threads and depending on if the main thread `join()`s on the exception-raising thread or not, the program can terminate or carry-on. Consider the snippet below:

```
# This example shows when a child thread throws an exception, the main thread
# is unaffected

faultyThread = Thread.new do

  puts("Child thread about to raise exception")
  # throw an exception
  raise "Ka Boom !"

  # Never gets printed
  puts("Child thread exiting")
end

sleep(1)
puts("Child thread status #{faultyThread.status}")
puts "Main thread exiting peacefully"
```

If we run the above snippet, the main thread is unaffected because it doesn't `join()` on the spawned thread. The child thread silently terminates on encountering an unhandled exception and the program carries-on. In the next snippet, we'll join on the child thread.

```
# This example shows when a child thread throws an exception, the main thread
# is unaffected

faultyThread = Thread.new do

  puts("Child thread about to raise exception")
  # throw an exception
  raise "Ka Boom !"

  # Never gets printed
  puts("Child thread exiting")
end

sleep(1)
puts("Child thread status #{faultyThread.status}")

faultyThread.join()

puts "Main thread exiting peacefully"
```



> In the above snippet, the exception is propagated to the main thread, which exits with a runtime error. We can choose to handle the exception in the main thread as follows:

```
# This example shows when a child thread throws an exception, the main thread
# is unaffected

faultyThread = Thread.new do

  puts("Child thread about to raise exception")
  # throw an exception
  raise "Ka Boom !"

  # Never gets printed
  puts("Child thread exiting")
end

sleep(1)
puts("Child thread status #{faultyThread.status}")

begin
  faultyThread.join()
rescue RuntimeError => e
  puts "Caught exception and swallowing it"
end
```

```
puts "Main thread exiting peacefully"
```

When we don't `join()` on a child thread but want the program to terminate if the child thread encounters an unhandled exception, we can set the `abort_on_exception` to true on the `Thread` class. When `abort_on_exception` is set to true, an exception in a child thread is re-raised in the main thread.

```
# This example shows when a child thread throws an exception, the program
# as a whole is aborted when Thread.abort_on_exception is set to true

Thread.abort_on_exception = true

faultyThread = Thread.new do

  puts("Child thread about to raise exception")
  # throw an exception
  raise "Ka Boom!"

  # Never gets printed
  puts("Child thread exiting")
end

sleep(1)

# Never gets printed
puts "Main thread exiting peacefully"
```

Since the exception from a child thread can be raised in the main thread only after the child thread has been spawned, we can wrap the code in the main thread following the child thread instantiation in a begin-rescue block to catch exceptions from the child thread. This is shown in the widget below:

```
# This example shows when a child thread throws an exception, the program
```

```
# as a whole is aborted when Thread.abort_on_exception is set to true

Thread.abort_on_exception = true


faultyThread = Thread.new do

  puts("Child thread about to raise exception")
  # throw an exception
  raise "Ka Boom!"

  # Never gets printed
  puts("Child thread exiting")
end

begin
  sleep(1)
  # Doesn't get printed because exception in child
  # thread is thrown when main thread is asleep
  puts "Main thread exiting peacefully"
rescue RuntimeError => e
  puts e.to_s
end

puts "The End"
```