Password Generator Notes

Requirements

This password generator will generate passwords as a random character string. These passwords would not be easy for a human to remember (a human would most likely write them down defeating the purpose of a password). However, these passwords would be suitable for storage into a password management program.

There is no definition of the minimum or maximum length for the passwords generated, it is assumed that this policy is enforced by the caller of the generatePassword operation.

The flags in the signature of generatePassword set whether a character category is “allowed”. An alternative password generator would set whether the generated password “must include at least one” from the character category. This could be simply implemented by calling generatePassword iteratively.

The signature of the generatePassword operation defines the return type to be a String. It is considered that this is not a good choice for a secure algorithm as there are concerns about String interning by the JVM which could leave passwords accessible in the JVM heap to attacker programs. It is notable that the Console.readPassword method in the JDK returns a character array and not a String.

Design & Implementation

The generatePassword implementation utilises a character array to build the password. Currently this has to be converted to a String at the point of return in order to satisfy the specified signature but it would be very easy to change this if the signature was changed (in accordance with the stated concerns above).

The PasswordGeneratorImpl class can be injected with an implementation of the RandomInteger interface in order to perform testing on the range of characters output by the generator. This has been utilised by the test suite. By default the PasswordGeneratorImpl is constructed with an implementation of RandomInteger which utilises the SecureRandom class from the JDK.

No statements have been made on the performance or thread safety of the implementation as these have not been specified in the requirements.

Various details of the implementation code could be improved by the addition of common jar dependencies, e.g. Guava for argument checking (Preconditions) and @VisibleForTesting, SLF4J as a preferable logging API to JUL and JUnit for construction of the test suite.

There may be suitable implementations for generatePassword in open source security libraries, e.g. “Bouncy Castle” but these have not been investigated for this exercise.

Deployment

For deployment this secure code should be packaged into a sealed and signed jar.

Management

I reviewed the specification on Friday evening and started to think about my approach to the problem. I wrote the code in two sessions of approximately two hours apiece on Saturday afternoon (before England v Wales) and on Saturday evening (before MOTD).

The code was developed in Eclipse (without FindBugs or Checkstyle plugins). Reference was made to Javadoc in the JDK but no reference was made to existing implementations of password generators.