Spring unchecked exceptions

Spring prefers unchecked exceptions because this way it gives to the developer possibility to choose to handle them or not. Developer is not enforced to handle them.

Best Practices for Exception Handling.

Consider below code:

```
public void consumeAndForgetAllExceptions() {
   try {
      //some code that throws exceptions
   } catch (Exception ex) {
      ex.printStacktrace();
   }
}
```

What is wrong with the code above?

Once an exception is thrown, normal program execution is suspended and control is transferred to the catch block. The catch block catches the exception and just suppresses it. Execution of the program continues after the catch block, as if nothing had happened.

How about the following?

```
public void someMethod() throws Exception{
}
```

This method is a blank one; it does not have any code in it. How can a blank method throw exceptions? Java does not stop you from doing this.

I want to know why spring handles only unchecked exceptions?

Personally I prefer unchecked exceptions declared in the throws cause. I hate having to catch exceptions when I'm not interested in them. I agree the spec needs to a few more exception types, but I disagree that they should be checked. Most frameworks rely on unchecked exceptions, not only Spring framework.

Best Practices for Designing the API

If the client can take some alternate action to recover from the exception, make it a checked exception.

If the client cannot do anything useful, then make the exception unchecked. By useful, I mean taking steps to recover from the exception and not just logging the exception.

The Java API has many unchecked exceptions, such as NullPointerException, IllegalArgumentException, and IllegalStateException. I prefer working with standard exceptions provided in Java rather than creating my own. They make my code easy to understand and avoid increasing the memory footprint of code.