Candidate Number: 2031B
Paper 1
Question 4

a. The program makes use of encapsulation by bundling data (the PushbackReader object defined on line 2) with methods which operate on that data (the read method on lines 5-11)

   The program makes use of abstraction by having several small functions (read on lines 5-11 and red on lines 15-19) which hide the complexity of the operations they perform from whichever other method calls them. They can be used as a "black box" where input is provided, output is gained, and the caller does not need to know how they work

   The program makes use of inheritance by allowing the Swapper class to extend Reader. This allows Swapper to inherit all the methods and attributes of a Reader object but without having to repeat the code (except for the Reader's read function which is overridden on lines 5-11 to give Swapper custom functionality)

   The program makes use of polymorphism when the Reader's read function is overridden by its child class Swapper on lines 5-11.

b. I would make the Swapper class a child class of PushbackReader rather than Reader. The read function could then call "this.unread" rather than "pushback.unread" since it itself will be a child class of PushbackReader. As such it will not need a reference to a PushbackReader object and no longer needs to be a decorator.

c. To implement the algorithm, the code of the Reader class itself does not need to be modified, and likewise with the PushbackReader class. Instead, these classes have their code extended (inheritance in the case of Reader and decoration in the case of PushbackReader). As such these classes are open for extension, but closed for modification.

d. I would use the strategy pattern. The class would instead be called CharPairFunctionApplier or something to that effect, and its constructor would accept an instance of an functional interface with a single unimplemented function "apply" whose arguments are a char array, and two indices, and which returns void.

Candidate Number: 2031B
Paper 1
Question 4

```
interface CharPairFunction {

	void apply(char[] cbuf, int a, int b);

}

class CharPairFunctionApplier extends Reader {
	private final PushbackReader pushBack;
	private final CharPairFunction func;

	CharPairFunctionApplier(PushbackReader p, CharPairFunction f) {
		pushBack = p;
		func = f;
	}

	@Override
	public int read(char[] cbuf, int off, int len) {
		int r = wrap.read(cbuf, off, len);
		if (r % 2 == 1) { pushBack.unread(cbuf, off + --r, 1); }
		for (int i = 0; i < r; i += 2) { func.apply(cbuf, i, i + 1);
}
		 return r;
	}
}

class Decryptor {
	static List<String> read(String fileName, CharPairFunction func)
{
		try (BufferedReader r = new BufferedReader(new
CharPairFunctionApplier(new PushbackReader(new FileReader(fileName)),
func))) {
			return readLines(r);
		 }
	}
}
```

e. If one wanted to create a class which decrypts images, e.g. ImageDecryptor, then they would have to modify the existing Swapper class or create an entirely new one, as the existing Swapper class only works on Strings. It is not possible to add this functionality simply extending existing functionality. This means that the class hierarchy would fail to reflect the conceptual hierarchy of what each class represents (e.g., ImageSwapper would not be a child class of Swapper like one might expect, and likewise ImageDecryptor of Decryptor).