SWEA Iteration 6: Compositional Design Principles

`<<Group name>>`
Computer Science, University of Aarhus
8200 Århus N, Denmark
<<Names>>

<<Date>>

1 Private Interfaces and ISP

1.1 Private Interface Design

[Describe your improved design using private interfaces for internal abstractions like Game, Card, or Hero]

1.2 Refactoring

[Show code before the refactoring; and next the refactored code based upon role interfaces.]

2 AbstractFactory

2.1 Design and UML

[Include a UML diagram that shows the design of HotStone with emphasis on the use of Abstract Factory. Please, do not draw all the association lines to all concrete Products/delegates as this will make the diagram a complete mess. (A good quality picture of a hand-drawn UML diagram is OK.)]

2.2 Configuring ZetaStone

[Write the full path of the the ConcreteFactory that configures HotStone for the ZetaStone.]

[Include screenshot/contents of the ConcreteFactory that configures HotStone for the ZetaStone variant.]

3 SemiStone

3.1 Configuration Table

[Fill in Table 1., similar to the table from FRS §17.2]

Table 1: HotStone configurations

	Variability points		
Product	**Mana Prod.**	**Winning**	**...**
AlphaStone	3 every round	Findus/round 4	-
BetaStone	+1 pr round	Defeat opponent	-
GammaStone	-	-	-
...	-	-	
SemiStone	-	-	-

3.2 SemiStone Code Configuration

The SemiStone variant is configured in our code like this ...

[Provide production code fragment(s) that show the code that configure the GameImpl for the SemiStone variant]

The design of the HotStone system, with emphasis on the SemiStone variant, is shown by the following UML diagram:

[Include the UML diagram of all interfaces and classes in HotStone related to variant handling—but show only the associations between the SemiStone abstract factory implementation to its products, *not* all the other associations between concrete factory classes.]

4 Parametric 'getWinner()'

The method getWinner() in GameImpl would look like this if a purely parametric design had been employed as variant handling technique in the HotStone code:

[Provide (pseudo) code fragment(s) the show how a parametric design could be implemented]

5 Polymorphic ZetaStone

The ZetaStone design would look like this if a purely polymorphic design had been employed as variant handling technique:

[UML diagram of the interfaces and classes involved in a purely polymorphic design of ZetaStone]

The actual getWinner() method in the ZetaStone subclass would look something like this

[(Pseudo-)Code fragment outlining the getWinner() method]

6 Backlog

- …
- …