Note: these are not model answers. They are notes on how the paper was done, which I hope will be useful to students looking back on this exam and people revising in future.

QUESTION 1.

1a) Bookwork: the extent to which the parts of the design depend on one another.

1b) Bookwork: decreasing coupling makes it easier (hence cheaper, quicker) to understand, test and debug.

1c) Most people got that more associations implied more coupling. Few people said that other kinds of arrows, especially generalizations, also indicate coupling: a class is strongly coupled to its superclass (you're dependent on stuff you inherit!) Ideally, your answer also recognised that it's arrows *out* of a class that indicate it is coupled to something.

1d) basic UML: it was a bit shocking that some people got this wrong. I accepted either the unfilled (aggregation) or filled (composition) diamond, which represent different kinds of "part" relationship.

1e) DIP is all about reducing the amount of stuff parts of a program depend on to the bare minimum (the interface they provide/require), so this was the answer expected. If you named a different SOLID element, I read your justification carefully and you may have got partial credit for it, but DIP is very much the best answer and most people gave it. For example, some people picked SRP, but that's really the opposite: it leads to highly cohesive classes, but lots of them, so higher coupling by most measures.

1f) Example of a fully correct answer:

```
context Customer inv:
self.active implies self.account -> size() > 0
and
self.account -> size() > 0 implies self.active
```

1g) Bookwork. Internal: hosted in a general purpose language. External: standalone. An internal language is, for example, typically easier to implement and more expressive (you only needed to give *one* advantage for the mark). An external language is, for example, typically simpler and easier to understand.

1h) The two modes are checkonly (checking consistency) and enforce (changing one model to restore consistency). The part about why both modes need access to both models was actually easier to answer correctly than I had intended - the point I was hoping you'd make is that the normal situation is that each model contains some information not contained in the other, so a bidirectional transformation isn't just a pair of one-way functions from one model set to the other, but in fact, many of you correctly made the point that enforce mode is only supposed to change anything if the models are not already consistent, and needs access to both models to check that. Most people got full marks for this part.

1i) It was surprising that even immediately after a question that had you thinking about how one bidirectional transformation has several modes, few people made the point that if you implement a bidirectional transformation in Java you will need three Java programs (or methods), one for checking, and one for restoring consistency in each direction.

QUESTION 2.

Some very strong essays were written, thoughtfully synthesising things said in the sources and elsewhere in the course. Where marks were lower, it was usually because the essay did not show evidence of familiarity with the readings. A few essays did not even show understanding that "model driven development" means more than just "development that involves drawing UML diagrams", and these got little credit.

QUESTION 3.

a) Bookwork. Most people could explain what a pattern is. Note that there were three marks so you needed a pretty full description. The commonest way to lose a mark was not to say something that indicated that a pattern is a named, structured description of a solution.

b) Bookwork again.

c) The answer I was looking for was State, for which this is a pretty classic application. Some people gave Strategy, instead; sensibly applied this got partial credit, as the structure is essentially the same. Where marks were lost it was usually through confusion over where the operations should be placed. It is surprising to have, for example, a cook() method in the Cooking class that isn't in the common superclass, because that means it can't be called from the context without a runtime type check on the state object, which would defeat the purpose of using State - so unless you carefully explained why you placed the operations as you did, you probably lost a mark if you did that.

d) This exercise was very similar to what we did with the Paleo example in the course. There were various ways to get it right - interesting issues included how you handled stuff that wasn't the original ingredients but was combined or partially cooked ingredients, like the scrambled egg, and how you handled the sequencing of steps of the recipe. If you didn't think about such issues you may have lost a small number of marks. Where people lost a lot of marks, it was usually at the more basic level, e.g., not seeming to know what a "metamodel" was.

e) There were lots of sensible things said here that got marks; the most commonly mentioned were how the robot was going to judge things like "fairly solid", and what should happen in case of error - several people correctly pointed out that there were safety issues to consider.