CHAPTER V

POLYMORPHIC INTERACTION PATTERNS

5.1 Polymorphic Interaction and Testing

To thoroughly test a polymorphic interaction, objects of subclasses are substituted to the polymorphic entity during test; as a consequence, each test case must be designed to execute a particular subclass. The previous chapter describes how to select subclasses for creating test scenarios. This chapter discusses how to create a test case to achieve a particular test scenario. In order to test a particular subclass in place of its superclass, how to substitute an object of the subclass to the polymorphic entity must be known. This chapter deals with this topic by defining 3 patterns of polymorphic assignment. Each pattern has a different way of how to control a polymorphic assignment.

5.1.1 Polymorphic Interaction

Polymorphism could only happen when there is a polymorphic assignment in an interaction. This is the basic requirement of a polymorphic interaction. In this research, patterns of polymorphic assignments are used for recognizing polymorphic interactions. The next section discusses about the patterns of polymorphic assignments.

5.1.2 Non-Polymorphic Interaction

Before discussion of the patterns of polymorphic assignments, the character of non-polymorphic interactions must be discussed first. It is important that this type of interaction is not confused with polymorphic interactions. For non-polymorphic interactions, polymorphism never happens, even though it may look like it could. The basic principle of polymorphic interaction is that there must be a polymorphic assignment in the interaction. An interaction without a polymorphic assignment is not a polymorphic interaction, although an entity in the interaction has one or more subclasses.

The UML class diagram in figure 5.1 is used for describing non-polymorphic interaction and also the patterns of polymorphic assignments following in the subsequent section. Class A has 3 subclasses. For an object of class A in an interaction, objects of the subclasses should be tested in the place of the object of class A, if the interaction is a polymorphic interaction.

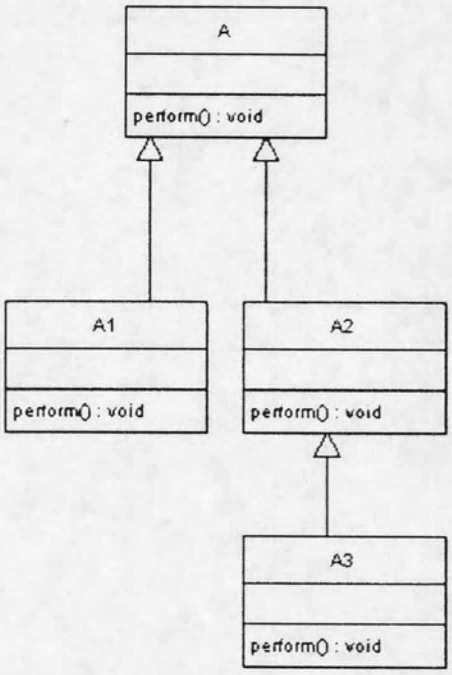

Figure 5.1 Example of a UML class diagram

The interaction illustrated in the UML sequence diagram in figure 5.2 seems to fit the requirement. However, it does not contain a polymorphic assignment. The object of class A is created directly through its constructor in the interaction. It is not possible to substitute the object with an object of one of its subclasses. This is not a polymorphic assignment; as a result, the interaction is not a polymorphic interaction.

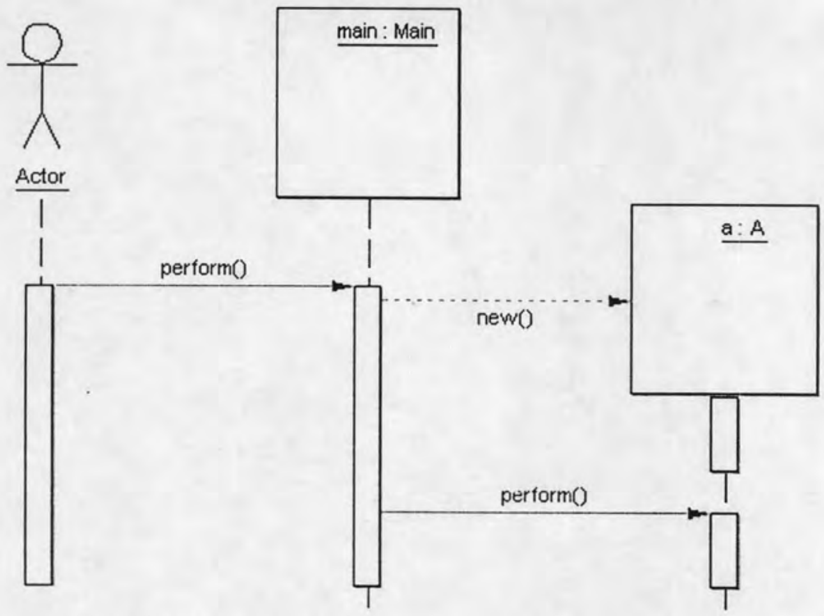

Figure 5.2 Example of an interaction without a polymorphic assignment

5.2 The Patterns

As stated earlier, patterns of polymorphic interactions are identified with patterns of polymorphic assignments. In this section, 3 patterns of polymorphic assignments along with their ways of controlling assignment are discussed. For each pattern, the characteristic that identifies the pattern is explained, and then how to control the assignment to assign a particular subclass is shown. Understanding these patterns allow testers to learn how to create test cases for polymorphic interactions which falls into one of these patterns.

Note that the class diagram in figure 5.1 is used throughout this section.

5.2.1 Simple Polymorphic Assignment

Simple polymorphic assignment is a polymorphic assignment pattern where the polymorphic entity is assigned as one of the arguments of the interaction. Figure 5.3 shows an example of Simple polymorphic assignment. Notice that the entity "a", which is a polymorphic entity, is assigned from the argument passed from the actor of the interaction.

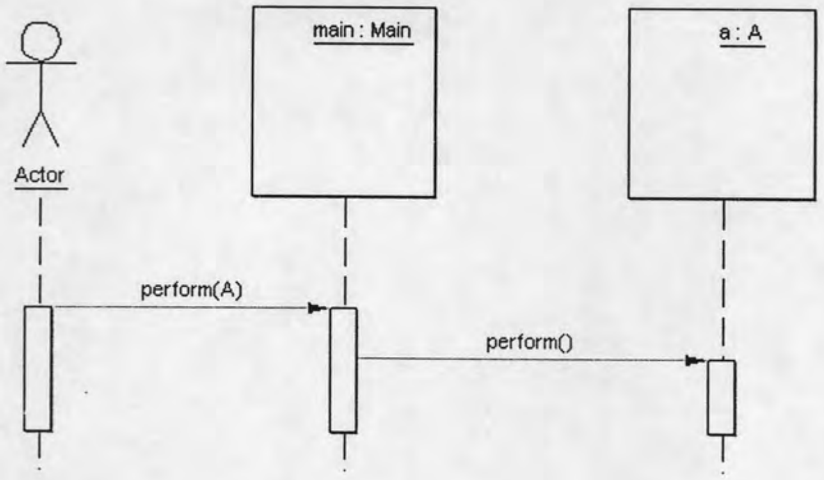

Figure 5.3 Example of Simple polymorphic assignment

For this type of polymorphic assignment, the assignment is controlled directly by the actor. If an object of a particular subclass of class A is required to substitute the entity a, the object is passed to the interaction as an argument. This pattern can be considered as the easiest to control among the three of them.

Test cases for a polymorphic interaction with this pattern of polymorphic assignment are created by having objects of required classes passed to the interaction as test input arguments. Although it is quite simple to create such test cases, test execution could be troublesome, since there is no way to tell how an object of a particular class under test should be created. This subject is beyond the scope of the research.

5.2.2 Parameter-Influenced Polymorphic Assignment

Parameter-influenced polymorphic assignment is an assignment which is dependent on one or more parameters of the interaction. It is similar to Simple polymorphic assignment, but the polymorphic entity is not assigned from the argument directly. Figure 5.4 shows an example of a UML sequence diagram of an interaction which contains Parameter-influenced polymorphic assignment.

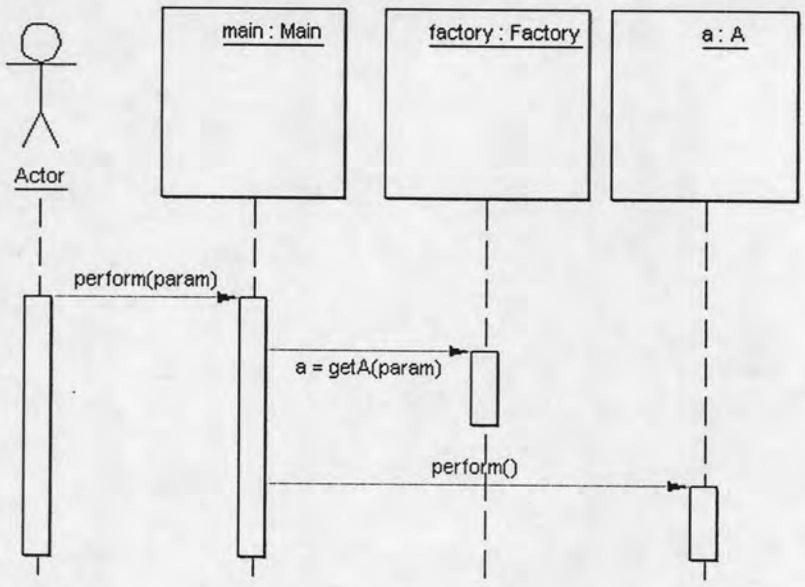

Figure 5.4 Example of Parameter-influenced polymorphic assignment

From the interaction in figure 5.4, the polymorphic entity "a" is assigned from the result of a call to the method "getA" on the factory object. The method takes an argument named "param" whose value is taken from the argument of the interaction of the same name. It is possible that the factory method would return an object of one of subclasses of class A, and the parameter of the method determines this behavior. One may argue that this cannot be identified by just the syntactic structure of the interaction. The semantic of the interaction must be taken into account to accomplish the recognition. This is discussed in detail in chapter 7 when an extension to UML sequence diagrams for identifying this pattern is discussed.

It is obvious that controlling this type of polymorphic assignment is to control the argument of the interaction. The problem is which arguments are to be controlled, and which values/objects are to be applied to the arguments. In addition to identifying which arguments influence the polymorphic assignment, chapter 7 also explains how to specify how arguments influence the polymorphic assignment in UML sequence diagrams using an extension proposed by this research.

5.2.3 Configuration-Influenced Polymorphic Assignment

Configuration-influenced polymorphic assignment is similar to Parameter-influenced polymorphic assignment in that the polymorphic entity is not directly assigned. It is influenced by one or more factors. This time it is influenced by factors which are not the arguments of the interaction. They are factors that are configuration like, e.g. object states, environment, data in data stores etc. An example of a UML sequence diagram of an interaction containing Configuration-influenced polymorphic assignment is shown in figure 5.5

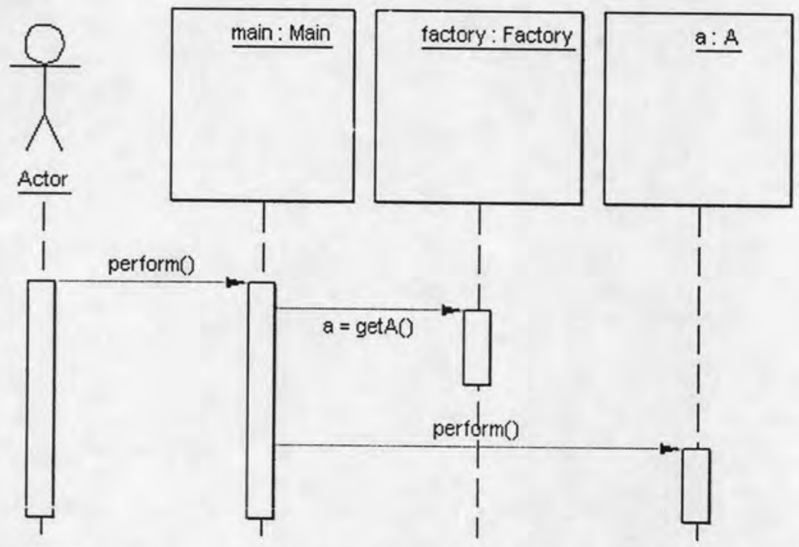

Figure 5.5 Example of Configuration-influenced polymorphic assignment

The interaction looks very similar to the interaction in figure 5.4. However, notice the difference in the factory method. The factory method in the interaction in figure 5.5 takes no argument; therefore, the behavior of the method does not rely on any argument. It may depend on object state, possibly the state of the factory object, or other external factors like data from data stores. This pattern is more implicit than Parameter-influenced polymorphic assignment. Identifying this pattern relies more on semantic information from the interaction.

Controlling this type of polymorphic assignment requires accessibility to the factors which influence the assignment. This is, however, the basic requirement of testability [8]. In chapter 7, the way to identify and control this type of polymorphic assignments in UML sequence diagrams is discussed. Similar to Parameter-influenced polymorphic interaction, this requires the help from an extension to UML sequence diagram.

5.3 Discussion

5.3.1 Variations

Discussed in the previous section are just the usual forms of the patterns of polymorphic assignments. It is possible that a polymorphic assignment in an interaction in real world may not look exactly like the forms presented in the previous section. The interaction in the UML sequence diagram in figure 5.6 is an example.

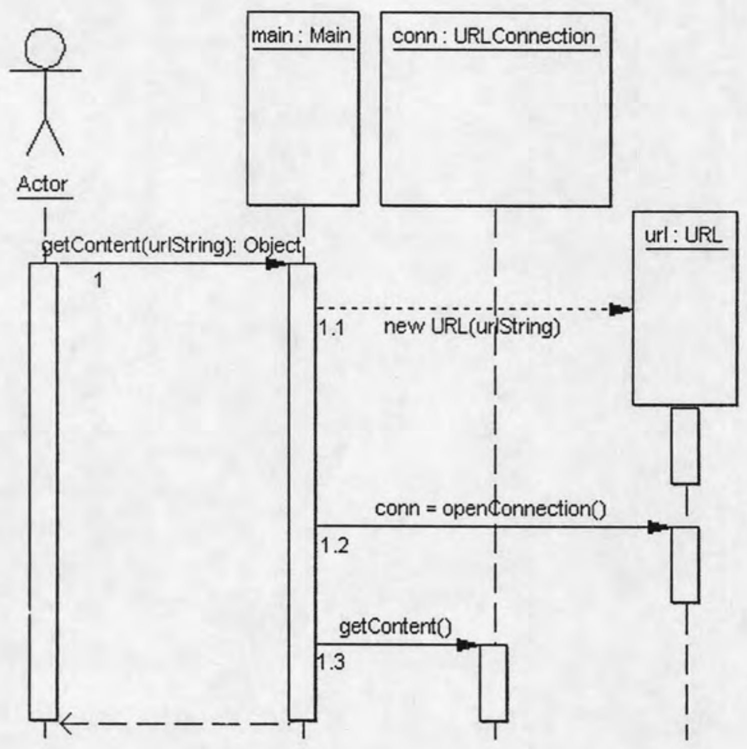

Figure 5.6 Example of a variation of polymorphic assignment patterns

The interaction in figure 5.6 expresses the way a Java program uses an object of URL class to create a connection to a particular resource described by a URL string. The connection is represented by an object of class URLConnection which has several subclasses, each representing connection to a particular protocol (e.g. HTTP, FTP etc.). The polymorphic assignment is at the step 1.2 in the interaction where a call is made to the URL object to get an object of class URLConnection. This may look like Configuration-influenced polymorphic assignment, since the method "openConnection", which is in the role of factory method, takes no argument. However, this is not the case. The method "openConnection" returns an object of one of the subclasses of URLConnection according to the protocol of URL. This is determined by the URL string of the resource represented by the URL object. The URL string must be supplied as an argument to the constructor of class URL when creating a URL object as shown at step 1.1 in the interaction. Consequently, the polymorphic assignment is essentially a Parameter-influenced polymorphic interaction.

This is just an example of how the patterns may have variations. It is important that the patterns are also recognized based on their semantic as well as their syntactic structures. It is not even remotely possible to identify all possible variations of each pattern; as a consequence, recognition of polymorphic assignments may sometimes require human decision.

5.3.2 Combination of Patterns

Another possible situation is when there is more than one polymorphic entity in an interaction. In addition, it is possible that different forms of polymorphic assignments happen in the same interaction. The patterns in this chapter are still applicable in this case, although the issue of combination is not discussed in detail. However, in the previous chapter the adequacy criteria for testing polymorphic interaction already address the situation where more than one polymorphic entity is involved. Identifying polymorphic assignment patterns for an interaction with more than one polymorphic entity is practically the same as for an interaction with only one polymorphic entity.

5.4 Examples

In this section, examples of patterns discussed earlier are illustrated in real world problem context to show how these patterns happen. Suppose a software system for a store is discussed. Figure 5.7 shows a UML class diagram of class Product and its subclasses. The diagram shows an inheritance hierarchy which is used for the discussion of this example. Notice that class Product and CD, which have their names italicized, are abstract classes.

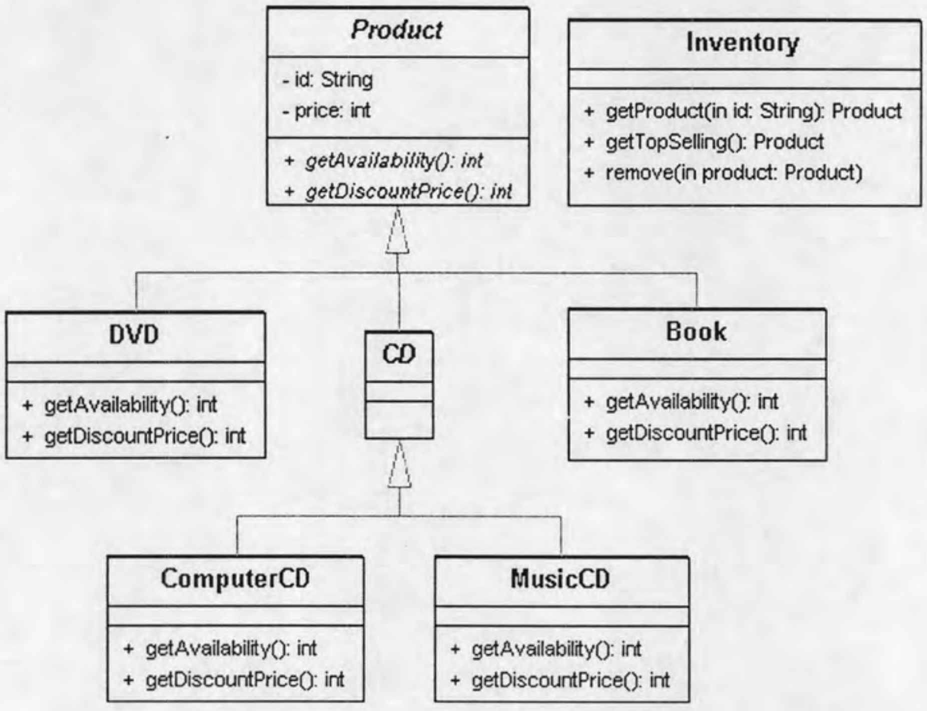
Product
- id: String
- price: int
+ *getAvailability(): int*
+ *getDiscountPrice(): int*
Inventory
+ getProduct(in id: String): Product
+ getTopSelling(): Product
+ remove(in product: Product)
DVD
+ getAvailability(): int
+ getDiscountPrice(): int
CD
Book
+ getAvailability(): int
+ getDiscountPrice(): int
ComputerCD
+ getAvailability(): int
+ getDiscountPrice(): int
MusicCD
+ getAvailability(): int
+ getDiscountPrice(): int

Figure 5.7 UML class diagram for store example

Shown in figure 5.8 is a UML sequence diagram of an interaction which illustrates how a product is purchased. The actor calls the method "purchase" on class Main passing an object representing a product to be purchased. The main object then proceeds to call a method on the inventory object to remove the product from the inventory and then call a method on the product object to get the discount price of the product respectively. The polymorphic entity is the product object, since it can be substituted with an object of one of its subclasses. The product object is assigned directly from the argument of the interaction; therefore, the polymorphic assignment is Simple polymorphic assignment.

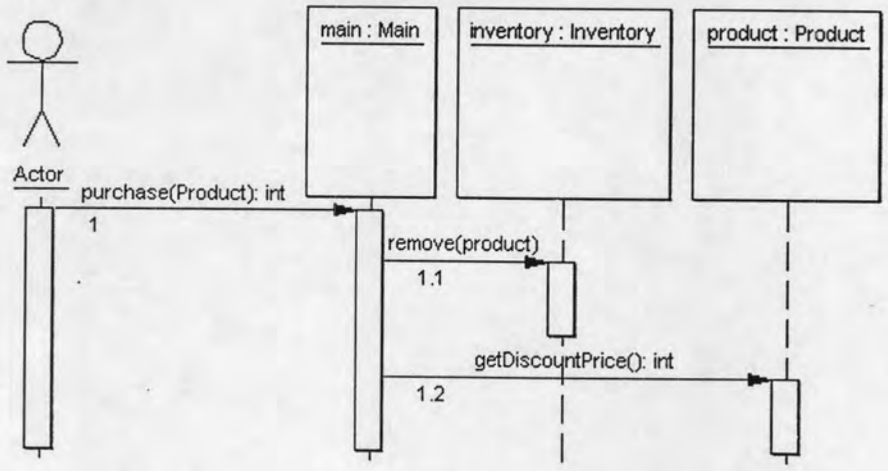

Figure 5.8 UML sequence diagram for purchasing product

Testing the interaction in figure 5.8 requires substitution of the product object. Since the interaction contains Simple polymorphic assignment, this can be achieved by simply passed objects of different subclasses to the interaction. Suppose that Inheritance coverage is selected as the adequacy criterion; as a consequence, it is required that all subclasses must be tested in the place of the polymorphic entity, as a result, all concrete classes from the inheritance hierarchy in figure 5.7 must be tested. An example of test cases for the interaction in figure 5.8 is shown in table 5.1. Note that the test cases are not complete. They just illustrate how the interaction should be tested.

Table 5.1 Test cases for the interaction in figure 5.8

No.	Test Input
1	An instance of Book
2	An instance of ComputerCD
3	An instance of MusicCD
4	An instance of DVD

Figure 5.9 shows another example of an interaction. It shows how availability of a particular product can be checked. The actor calls the method "checkAvailability" on

the main osbject passing the id of the product to be checked. The main object then calls the finder object to obtain the product object of the given id and finally calls a method on the obtained product object to get availability.

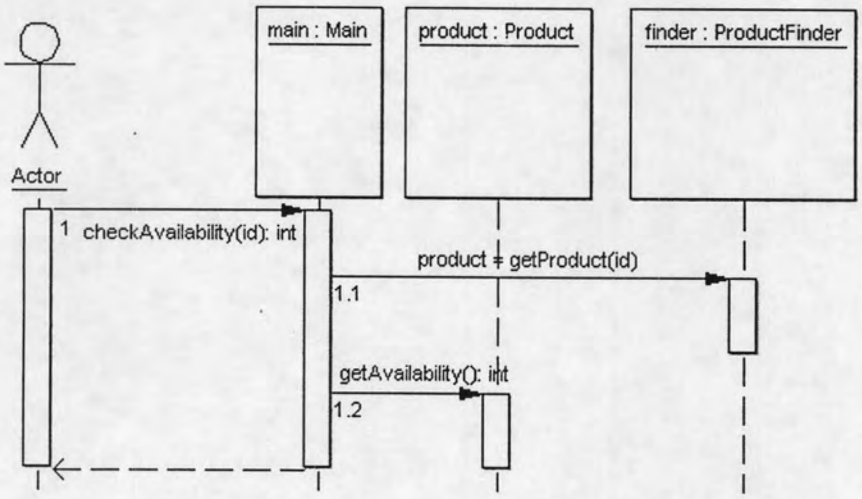

Figure 5.9 UML sequence diagram for checking product availability

From the interaction, the finder object is in the role of a factory. It returns an object of various product types depending on the given id. Since the id is supplied from the actor as the argument of the interaction, the polymorphic assignment is Parameter-influenced polymorphic assignment.

The interaction in figure 5.9 requires its test cases to be created by varying the product id which is the argument of the interaction. Table 5.2 shows an example of test cases for the interaction.

Table 5.2 Test cases for the interaction in figure 5.9

No.	Test Input (id)
1	Id of a book
2	Id of a computer CD
3	Id of a music CD
4	Id of a DVD

The interaction in figure 5.10 shows how the availability of the top selling product can be checked. The actor calls the method "getTopSellingAvailability" on the main object. The main object then obtains a product object from the finder object and gets availability from the obtained product object. This looks quite similar to the previous interaction where the product object, as a polymorphic entity, is obtained from the finder object, as a factory.

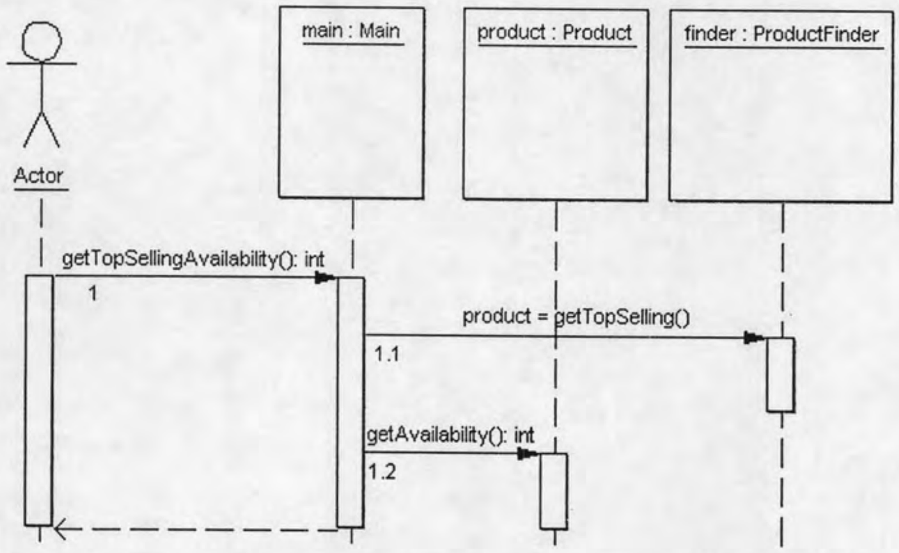

Figure 5.10 UML sequence diagram for checking availability of the top selling product

However, this interaction is not Parameter-influenced polymorphic assignment like the previous interaction. The method "getTopSelling" takes no argument here; it is not influenced by any arguments in the interaction. In this case, the top selling product is determined by the finder object possibly by checking with the sales history records in a data stores. This is considered as an environmental factor external to the interaction; hence, the polymorphic assignment is Configuration-influenced polymorphic assignment.

Testing the interaction then requires that the top selling product is set to different types of product. How to exactly do so depends on specific implementation of the interaction, the finder object in particular. For example, the finder object may perform summation on sales records to determine the top selling product. It is also possible that there is a place in the data store that keeps the id of the top selling product which is updated periodically based on statistical methodology. Different implementation requires different ways to be tested. Due to insufficient information, test cases may not identify exactly how to test the implementation. An example of test cases for the interaction is shown in table 5.3.

Table 5.3 Test cases for the interaction in figure 5.10

No.	Test Set-up
1.	Set the top selling product to be a book
2.	Set the top selling product to be a computer CD
3.	Set the top selling product to be a music CD
4.	Set the top selling product to be a DVD