cụ thể [6]. Vào cuối những năm 1960, Floyd, Hoare và Naur đề xuất kỹ thuật tiền đề để chứng minh sự nhất quán giữa các chương trình tuần tự và các mô tả của chúng, được gọi là các đặc tả [9][10][11]. Sự quan tâm về các đặc tả hình thức và các ứng dụng đa dạng của chúng trong lĩnh vực công nghệ phần mềm đã phát triển liên tục cho đến hiện nay [12][13].

Một trong những kết quả mới và có tính thực tiễn nhất trong hướng tiếp cận này là việc đưa ra các "thoả thuận" giữa yêu cầu của một hàm và hiện thực của hàm đó [7]. Như đã giới thiệu trong Phần 1, một "thoả thuận" được biểu diễn thông qua các tiền điều kiện và hậu điều kiện. Ví dụ, một "thỏa thuận" cho phương thức *sqrt* (nhận vào một số và trả về căn bậc hai của nó) được biểu diễn bằng ngôn ngữ JML (*Java Modeling Language*) [14] như mô tả trong Hình 1. Trong ví dụ này, mệnh đề *require* thể hiện tiền điều kiện, là biến *x* truyền vào phải lớn hơn hoặc bằng *0*. Mệnh đề *ensure* thể hiện hậu điều kiện. Ở ví dụ này, giá trị trả về (*\result*) phải thỏa điều kiện là bình phương của nó phải bằng lại giá trị của biến *x* truyền vào.

```
//@ requires x >= 0.0;
//@ ensures \result * \result = x
public static double sqrt(double
x)
{ /*...*/ }
```

Hình 1: "Thỏa thuận" của phương thức *sqrt* được biểu diễn bằng ngôn ngữ JML

2.2 Các ngôn ngữ đặc tả hình thức theo hướng tiếp cận "thoả thuận"

2.2.1 Ngôn ngữ JML

JML (Java Modeling Language) [14] là ngôn ngữ đặc tả hành vi, được sử dụng để đặc tả hành vi của các mô đun trong ngôn ngữ Java. Nó là sự kết hợp phương pháp "thỏa thuận" với phương pháp đặc tả dựa trên mô hình của ngôn ngữ Larch [15]. JML sử dụng logic Hoare [16] kết hợp với việc đặc tả các tiền/ hậu điều kiện (pre-/post-condition) và các bất biến (invariant) của các thuộc tính, các cấu trúc.

Các đặc tả JML được thêm vào mã Java dưới dạng các chú giải (annotation) bên trong các chú thích (comment). Các chú thích trong ngôn ngữ Java được hiểu như là các chú giải JML khi nó bắt đầu bằng ký tự "@". Ví dụ trong Hình 2 minh họa việc sử dụng JML đặc tả cho lớp *Purse*. Trong đó, trường *balance* được đặc tả với bất biến (***invariant***) là *balance* luôn trong đoạn từ 0 đến *MAX_BALANCE*. Trường *pin* là một mảng thuộc kiểu ***byte*** với bất biến là *pin* có đúng 4 phần tử và các phần tử chỉ có giá trị từ 0 đến 9.

```
public class Purse {
  final int MAX_BALANCE;
  int balance;
  //@ invariant 0 <= balance &&
            balance <= MAX_BALANCE;

  byte[] pin;
  /*@ invariant pin != null &&
        pin.length == 4 &&
       (\forall int i; 0<=i&&i<4;
         0<=pin[i]&&pin[i]<=9);
*/

  /*@ requires amount >= 0;
   @ assignable balance;
   @ ensures
      balance==\old(balance)-
amount
       && \result == balance;
   @ signals (PurseException)
      balance == \old(balance);
*/
   int debit(int amount) throws

PurseException;
   }
```

Hình 2: Ví dụ minh họa về đặc tả JML

Phương thức *debit* nhận vào đối số *amount*. Phương thức này sẽ trừ *amount* vào *balance* và trả về giá trị của *balance* sau khi trừ trong trường hợp *balance* lớn hơn hoặc bằng *amount*. Trong trường hợp *balance* nhỏ hơn *amount*, một ngoại lệ *PurseException* sẽ được trả về. Tiền điều kiện (***requires***) của phương thức này là *amount* phải lớn hơn hoặc bằng 0; hậu điều kiện (***ensures***) là *balance* bị trừ đi một lượng *amount* và trả về kết quả sau khi trừ. Phương thức này còn có một hậu điều kiện nữa cho trường hợp phương thức trả về ngoại lệ (***signals***).

2.2.2 Spec#

Spec# [17] là một ngôn ngữ hình thức cho các "thỏa thuận" API (có sự ảnh hưởng từ JML), nó là sự mở rộng của ngôn ngữ C# với việc bổ sung các tiền/ hậu điều kiện và bất biến cho các đối tượng. Spec# là ngôn ngữ đặc tả cho phép kiểm tra động cũng như kiểm tra tĩnh chương trình. Hệ thống Spec# cung cấp một kiến trúc hoàn chỉnh, bao gồm trình biên dịch, thư viện hỗ trợ và các công cụ phát triển. Các đặc tả của Spec# cũng được biên dịch thành một phần của chương trình thực thi, trong đó