Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Compiladores e Intérpretes

Semántica de MiniJava- Parte I - Declaraciones

Segundo Cuatrimestre de 2025

1 Introducción

Este documento una descripción (no exhaustiva) de la semántica del lenguaje de programación MiniJava. Como se ha visto a lo largo de la materia, MiniJava es en gran parte un subconjunto de Java y, por lo tanto, la mayoría de las sentencias en MiniJava tienen la misma semántica que en Java. La principal diferencia entre los lenguajes está en las sentencias y operadores que estos permiten. MiniJava elimina muchas características de Java tales como hilos, excepciones, variables de clase, métodos abstractos, arreglos, operaciones de punto flotante, etc.. Además, MiniJava restringe el uso de algunas características como la posibilidad de declarar, en una clase, varios métodos con el mismo nombre y diferente cantidad de parámetros. Este documento proporciona guías para determinar qué declaraciones de un programa MiniJava sintácticamente válido son semánticamente válidas (*i.e.*, compilan sin errores).

2 Consideraciones Generales

Cada declaración de una clase, método, variable de instancia o parámetro formal asocia a la entidad un nombre y determina algunas de sus características asociadas. Los programas MiniJava luego podrán hacer referencia a una entidad particular a través de su nombre. Por lo tanto, el compilador de MiniJava deberá generar y mantener un símbolo para cada entidad declarada. Estos símbolos deberán organizarse en tablas de símbolos, asociadas a clases y/o métodos, ya que diferentes entidades tendrán diferente alcance. Por ejemplo, un parámetro de un método `met1` sólo podrá utilizarse dentro del cuerpo de `met1`, mientras que un atributo en una clase `c1` podrá utilizarse en cualquier método no estático de `c1`. De esta manera, el objetivo de la tabla de símbolos es dar la posibilidad de obtener las características de una entidad a partir de su nombre en el contexto adecuado. Por ejemplo, si `a` es un parámetro de un método `met2`, cuando se hace referencia a `a` en el cuerpo de `met2` la tabla de símbolos deberá brindar todas las características de ese parámetro (*e.g.*, tipo, nombre, posición).

Una posible forma de organizar las tablas de símbolos es la siguiente. En primer lugar, todas las entidades de tipo clase se almacenan en la *tabla de símbolos de clase*. Luego, cada elemento de esa tabla deberá contener una (o varias) tabla(s) con métodos, constructores y variables de instancia. A su vez, cada método y cada constructor tendrá asociada una tabla propia para almacenar los parámetros y variables locales del método o constructor.

2.1 Declaraciones de clase

El nivel superior de un programa MiniJava es una secuencia de una o más declaraciones de clase. En la declaración de una clase concreta se declaran todas sus atributos, métodos con cuerpo y a lo sumo un constructor. A diferencia las clases abstractas, pueden tener métodos sin cuerpo y no tienen constructor.

Todas las clases declaradas son globalmente visibles. Aun así, no pueden declararse dos clases con el mismo nombre. De ocurrir esta situación se reportará un error semántico. Note que, de esta manera, no podrán declararse clases concretas llamadas `Object`, `String` o `System`.

Entonces, basicamente, las declaraciones de clase de MiniJava son como sus homónimas en Java, con la diferencia de que permiten menos combinaciones de elementos o sus modificadores son mas restrictivos.

2.1.1 Herencia

Similar a como es en Java, en MiniJava es posible modelar herencia simple entre clases. Al igual que en Java es posible establer herencia entre clases concretas y abstractas. Las clases abstractas pueden tener o heredar métodos abstractos, mientras que las clases concretas tienen que tener todos sus métodos concretos. Esto último obliga a que una una clase concreta que hereda de una abstracta tenga que redefinir todos los métodos abstractos heredados haciendolos concretos. La redefinicion de métodos tiene algunas restricciones adicionales a Java, para los detalles ver 2.3.

Las clases final y static no pueden ser heredadas y una clase abstracta no puede heredar explicitamente de una concreta.

En MiniJava toda clase tiene una superclase, con excepción de la clase predefinida `Object`. Si se declara una clase sin la palabra `extends`, entonces esa clase será por defecto una subclase directa de `Object` [1]. Si se explicita una clase de la cual se hereda esta debe estar declarada o ser predefinida. Además, al igual que Java, MiniJava controla que cualquier tipo de herencia no sufra de **circulararidad**.

2.1.2 Clases Predefinidas

Al igual que en Java, en MiniJava hay clases concretas predefinidas que pueden utilizarse sin la necesidad de ser previamente declaradas y definidas. Se cuenta con dos clases predefinidas:

- **Object**: La superclase de todas las clases de MiniJava (al estilo de la clase `java.lang.Object` de Java). A diferencia de java, en MiniJava, la clase **Object** solo posee el siguiente método estático:
  - `static void debugPrint(int i)` que imprime un entero por la salida estandar y finaliza la linea.

  Con este método cualquier clase del sistema (dado que todas heredan de **Object**) tiene la capacidad de imprimir por pantalla un numero sin la necesidad de requerir de System
- **String**:La clase predefinida para caracterizar los objetos de los literales string. En MiniJava, la clase String no posee métodos ni atributos.
- **System**: Contiene métodos útiles para realizar entrada/salida (al estilo de la clase `java.lang.System` de Java). A diferencia de `java.lang.System` esta clase sólo brinda acceso a los streams de entrada (System.in) y salida (System.out), pero lo hace de manera oculta proveyendo directamente los siguientes métodos:
  - `static int read()`: lee el próximo byte del stream de entrada estándar (originalmente en la clase `java.io.InputStream`).
  - `static void printB(boolean b)`: imprime un `boolean` por salida estándar (originalmente en la clase `java.io.PrintStream`).
  - `static void printC(char c)`: imprime un `char` por salida estándar (originalmente en la clase `java.io.PrintStream`).

[1] Notar que las clases abstractas sin herencia explicita, heredan de `Object`, siendo esta la unica forma de que una clase abstracta herede de una concreta. Esto debe ser tenido en cuenta al realizar el control correspondiente.

- `static void printI(int i)`: imprime un `int` por salida estándar (originalmente en la clase `java.io.PrintStream`).
- `static void printS(String s)`: imprime un `String` por salida estándar (originalmente en la clase `java.io.PrintStream`).
- `static void println()`: imprime un separador de línea por salida estándar, finalizando la línea actual (originalmente en la clase `java.io.PrintStream`).
- `static void printBln(boolean b)`: imprime un `boolean` por salida estándar y finaliza la línea actual (originalmente en la clase `java.io.PrintStream`).
- `static void printCln(char c)`: imprime un `char` por salida estándar y finaliza la línea actual (originalmente en la clase `java.io.PrintStream`).
- `static void printIln(int i)`: imprime un `int` por salida estándar y finaliza la línea actual (originalmente en la clase `java.io.PrintStream`).
- `static void printSln(String s)`: imprime un `String` por salida estándar y finaliza la línea actual (originalmente en la clase `java.io.PrintStream`).

Nótese que, a diferencia de Java, cada método de impresión tiene un nombre diferente, ya que en MiniJava una clase no puede tener más de un método con el mismo nombre.

2.2 Declaración de Variables de Instancia (Atributos)

Sigue la semántica de Java para las variables de instancia (atributos) en clases concretas. En una clase concreta no puede haber más de una variable de instancia en con el mismo nombre. La visibilidad para las variables sigue la semántica de Java para los atributos que no tienen visibilidad explicita (los asume como public). Por otra parte, al diferencia de Java, no es posible declarar una variable con el mismo nombre que una definida en una superclase. Finalmente, si una variable de instancia es declarada de tipo clase, esa clase tiene estar declarada.

2.3 Declaración/Definición de Métodos

A diferencia de Java, en MiniJava no pueden declarase dos métodos con el mismo nombre dentro de una misma clase. La visibilidad para los métodos sigue la semántica de métodos *public* en Java. Además, al igual que en Java, si el método es declarado con un retorno de tipo clase, esa clase tiene que estar declarada.

Como en java los métodos pueden tener modificadores. En MiniJava solo se permite usar uno entre static, final y abstract. La semantica las implicancias semanticas de cada uno de estos modificadores son las mismas que en Java, con algunas restricciones que se explican a lo largo de este documento.

Al igual que en Java, es posible declarar un método con el mismo nombre que un método definido en una superclase (en cuyo caso la subclase sobre-escribe el método de la superclase).Si una clase sobre-escribe un método, éste debe tener exactamente la misma cantidad y tipo de parámetros, el mismo tipo de resultado que el método sobre-escrito.

Los modificadores afectan tambien las sobreescrituras. Por un lado los métodos final y static no pueden ser redefinidos. Por el otro un método heredado abstract puede ser redefinido, pero en la clase descendiente tiene que tener cuerpo no vacio, no tienen que tener el modificador abstract y, opcionalmente, podría llegar a tener el modificador final.

Finalmente todos los tipos utilizados en la declaración de un método tienen que estar delcarados o ser predefinidos.

2.4 Constructores

Los constructores en MiniJava funcionan como en Java. La principal diferencia es que una clase concreta en MiniJava sólo puede tener declarado/definido un único constructor. Al igual que en Java,

en caso que no se declare ninguno, el compilador le asignará un constructor por defecto (sin argumentos y con cuerpo vacío) a todas las clases concretas (incluidas las predefinidas). Además si declara un constructor para una clase concreta debe tener el mismo nombre que la clase.

2.5 Declaración de Parámetros Formales

Los parámetros formales en las declaraciones de unidades siguen una semántica similar a la de Java. Un método no puede tener más de un parámetro con el mismo nombre. Finalmente, si un parámetro es declarado de tipo clase, esa clase tiene estar declarada.

3 Chequeos Semánticos

Los chequeos semánticos tienen dos objetivos. El primer objetivo es descartar programas inválidos, por ejemplo: un programa con una delcaracion `class A extends A;` debería ser rechazado porque tiene herencia circular. El segundo objetivo es recolectar información acerca de los tipos que será luego utilizada en la generación de código.

En las siguientes subsecciones se presentarán informalmente consideraciones necesarias para realizar el chequeo semántico de un programa MiniJava.

3.1 Chequeos Semánticos referentes a las Declaraciones

Básicamente, es necesario chequear que en toda declaración donde se utiliza un tipo no primitivo (ej: declaración de herencia, variables o métodos) el nombre haya sido declarado, es decir, que corresponda a una clase existente. Entonces, para determinar si el nombre utilizado es correcto, se consulta la tabla de símbolos de clases. Si la clase no está en esa tabla, esto quiere decir que no se ha declarado y por lo tanto hay un error semántico. Existen otros chequeos que deben efectuarse sobre las declaraciones, para mas detalles ver las secciones anteriores.

Estos controles suelen realizarse en la segunda pasada, como parte del *chequeo de declaraciones*, directamente sobre la tabla de símbolos. Es decir, para toda clase almacenada en la tabla de símbolos se controla si todos sus elementos fueron correctamente declarados. Aun así, es más simple realizar los controles relativos a los nombres repetidos mientras se crea la tabla de símbolos en el analizador sintáctico.

4 Consideraciones de Implementación del Chequeador de Declaraciones de MiniJava

Como se mencionó anteriormente, para realizar adecuadmente el análisis semántico de los programas MiniJava será necesario realizar dos pasadas. A continuación se mencionarán algunas consideraciones que pueden ser útiles para el diseño de cada una de estas pasadas.

4.1 Primera Pasada

En la primera pasada, mientras el analizador sintáctico reconoce el archivo de entrada, irá construyendo la tabla de símbolos. Si se sigue la forma sugerida en la Sección 2, entonces a medida que el analizador sintáctico procesa declaraciones de clases irá agregando clases a la *tabla de símbolos de clase*. Cuando procese las declaraciones de miembros de una clase (métodos y atributos) los agregará a las tablas de métodos y atributos de esa clase. Además, en particular, cuando procese las declaraciones de una unidad se agregará todos los parámetros a la tabla de parámetros de esa unidad. Siempre que se trate de agregar una entidad en la tabla de símbolos se chequeará si el nombre no esta repetido.

4.2 Segunda Pasada

Al terminar la primer pasada, si bien se generó por completo la tabla de símbolos, aún no se realizó ningún control semántico. En la segunda pasada se procederá a realizar dichos controles. Estos controles se denominan chequeo de declaraciones.

En el chequeo de declaraciones se realiza directamente sobre la tabla de símbolos utilizando las entradas para las clases, atributos, métodos y constructores, para ver si estas fueron correctamente declaradas. En esta etapa no se chequea el cuerpo de los métodos, sino que sólo se chequea que sus encabezados (parámetros, modificadores y tipo de retorno) estén correctamente declarados. En la Sección 3.1 se presenta información un poco más detallada de los chequeos sobre las declaraciones.

Además, en esta etapa se controlará si algún conjunto de clases sufre de herencia circular. Es decir, para toda clase deberá chequearse que no sea ancestra de sí misma. En esta pasada también se actualizarán las tablas de métodos y las tablas de variables de las clases en base a la relación de herencia, proceso que denominamos *consolidación*. En particular, en la consolidación, se deberán agregar todos los métodos y las variables que la clase hereda de sus ancestros, con excepción de aquellos que esta sobre-escribe. Finalmente, cuando se vea la generación de código en la *etapa 5*, se generan los *offsets* asociados a las variables de instancia de cada clase.