MiniJava programming language for CS 358

(Note that this is significantly different than Appel's version of MiniJava in the textbook.)

MiniJava is a subset of Java. Any MiniJava program should be able to be run as a Java program, provided that the RunMain and Lib classes are available. Many Java programs, however are not MiniJava programs. This document describes the restrictions that are placed on MiniJava, when compared to regular Java.

identifiers and literals

- identifiers (e.g., variable names)
  - may not start with '_'; may not contain '$'
- integer literals (e.g., 1234)
  - may not start with a 0, except for the single-character token, '0'
  - no suffixes, as in 1234L (to indicate *long*)
  - 2147483648 not allowed, even when prededed by a unary -
- string literals (e.g., "hello")
  - the only escape sequences that are supported are: \f, \n, \r, \t, \", \', \\
- character literals (e.g., 'X')
  - they have type *int*, not *char* as in Java
  - same escape sequences supported as for string literals

reserved words

The only reserved words allowed are the following

- class
  - may not be declared as public, only "plain"
- extends
- public
  - only allowed in method header--and method headers require it
- return
  - only allowed as the last statement of value-returning function
- boolean
- int
- void
- if
- else
- while
- break
  - labeled break statements are disallowed, as are the labels that they would refer to
- for
  - 'foreach' style loop (e.g., "for (String s : someArray)") is disllowed
- switch
  - switch-expression must have type int
  - switch-blocks must consist of zero or more clealy delimited groups, each beginning with one or more case-labels (including 'default:'),and ending with a 'break;'.
- case
- default
- new
  - objects may be created, as in "new MyClass()". No parameters may be passed.
  - arrays may be created, as in "new int[4]" or "new int [5][][]", but filled-in multidimensional arrays may not be directly created, as in "new int[5][2][7]"
- null
- true
- false
- this
- super
  - only allowed as part of method call
- instanceof (see below, under "operators")

unused reserved words

Constructs that use the following Java reserved words are not supported in MiniJava:

abstract, assert, byte, catch, char, const, continue, do, double, enum, final, finally, float, goto, implements, import, interface, long, native, package, private, protected, short, static, strictfp, synchronized, throw, throws, transient, try, volatile

To be consistent with Java, these reserved words may not be used as plain identifiers in MiniJava. You therefore may not declare a variable with the name "try".

grouping and separator characters

As in Java:

- `,` is used to separate method parameters
- `(` and `)` are used for grouping, for casts, and for parameters
- `{` and `}` are used to group statements
  - They are not used for array-initializers
- `:` is used for case-labels
  - It is not used for loop-labels or as part of a conditional expression

operators

- `=` — assignment
  - does not produce a value
- `++` — increment (both prefix and postfix)
  - does not produce a value; may only be applied to a plain variable
- `--` — decrement (both prefix and postfix)
  - does not produce a value; may only be applied to a plain variable
- `||` — boolean "or"
- `&&` — boolean "and"
- `!=` — "not equals"
- `==` — "equals"
- `<=` — "less than or equal"
- `>=` — "greater than or equal"
- `<` — "less than"
- `>` — "greater than"
- `+` — "plus"
  - unary: identity operator for int
  - binary: may not be applied to strings
- `-` — "minus"
  - unary: negation
  - binary: subtraction
- `*` — "times"

- `/` — "division"
- `%` — "remainder"
- `!` — unary boolean complement
- `instanceof`
  - only plain class types may be used, as in "exp instanceof MyClass"
- `( )` — cast (prefix)
  - only plain class types may be used, as in "(MyClass)exp"
- `[ ]` — array-element access
- `.` — instance variable access (as in "a.b") or method call (as in "a.f()")

unused operators

The following operators not supported:
- bitwise: `^ & | ~ << >> >>>`
- compound assignment: `+= -= *= /= %= ^= &= |= <<= >>= >>>=`
- conditional expression, which uses `?` and `:`

other restructions

- The entire program must be in one source file.
- An instance variables may not be named 'length'.
- Instance variables may not be declared as public.
  - they must be declared as "plain". They are, in effect, public, however, because all classes are in the same package.
- Local variables are required to have an intiailizer.
- Function overloading is not supported.
- No inner classes or anonymous classes.
- No generics.
- No finalizers.
- No C-style array declarations: must say "int[] x;" rather than "int x[];"
- Cannot declare multiple variables in same declaration (so no "int x, y;")
- No brace-enclosed array initializers, as in "int[]x = {1,2,3};"
- Constructors may not be explicitly declared, so the only the only constructor available for a class is the one that initializes instance variables to the default values of 0, false and null.

library

The Java API library is not supported. Two classes, Object and String, are partially supported
- Object: only methods supported are 'toString', 'equals' and 'hashCode'
- String: only methods supported are 'toString', 'equals', 'hashCode', 'length', 'charAt', 'compareTo', 'concat' and 'substring' (two-operand version)

Additionally, the following library classes have been defined:
- RunMain: contains only the "public static void main" method
  - its sole purpose is to start the program in a way that is consistent with normal Java
  - invokes "new Main().main();", and catches a handful of the common exceptions, printing an error message for each and then exiting
  - no constructor available for this class
- Lib: supports the following public methods:
  - `public String readLine();`
    Reads and returns a line of text from the keyboard. Analogous to readLine() in the BufferedReader class.
  - `public int readChar();`
    Reads a single character from the keyboard. Returns the ASCII code for the character read. For example, if a space character is read, the method will return the value 32.
  - `public int readInt();`
    Reads a sequence of digits from the keyboard (optionally preceded by some whitespace and/or a minus sign) returns the integer represented (in base 10) by that sequence of digits).
  - `public void printStr(String s);`
    Prints a string to the console screen.
  - `public void printInt(int n);`
    Prints an int (in base 10) to the console screen.
  - `public void printBool(boolean b);`
    Prints a boolean value to the console screen (either 'true' or 'false').
  - `public String intToString(int n);`
    Converts an int into a string (using base 10).
  - `public String intToChar(int n);`
    Converts an int to a one-character string containing the character with that ASCII code (more precisely its lowest eight bits).

None of the predefined classes may be redeclared.

Example MiniJava program

```
// Main class -- prompts user; prints primes
class Main extends Lib {

   // main method: prompts user and prints that many primes
   public void main() {

      // prompt user for count; read response from keyboard
      printStr("How many primes do you want? ");
      int num = readInt();

      // create ListPrimes object
      Primes p = new Primes();

      // print the specified number of primes
      int primeCount = 0; // # primes found so far
      for (int i = 0; primeCount < num; i++) {
         if (p.isPrime(i)) {
            // found a prime: print, and increment count
            printInt(i);
            printStr("\n");
            primeCount++;
         }
      }
   }
}

// Primes class - tests for primality
class Primes { // tells whether a number is prime

   // tests whether a number is prime
   public boolean isPrime(int n) {
      // return-value, initialize for case where n <= 1
      boolean rtnVal = false;

      // if n > 1, test all numbers in 2..sqrt(n) for
      // being a factor.  Return true iff no factor found
      if (n > 1) {
         rtnVal = true; // have not yet found a factor
         int limit = sqrt(n); // limit: square root of value
         // test numbers in range for being a factor
         for (int i = 2; i <= limit; i++) {
            if (n % i == 0) {
               // found factor; set return-val to false; break
               // out of loop
               rtnVal = false;
               break;
            }
         }
      }
      return rtnVal;
   }

   // integer square root method: computes the floor of the square root
   // of an integer if zero/positive; returns zero if negative
   // determines square root using binary search
   public int sqrt(int n) {
      // return value, set up with default value for n <= 0
      int rtnVal = 0;

      // if n > 0, use binary search to narrow in on square root
      if (n > 0) {
         // min and max possible values for answer
         int min = 1;
         int max = 65535;

         // binary search loop; keep going until min and max meet
         while (min < max) {
            // compute midpoint
            int middle = (min+max+1)/2;

            // if square of midpoint too large, reduce max; otherwise
            // decrease min
            if (middle*middle > n) {
               max = middle-1;
            }
            else {
               min = middle;
            }
         }

         // min and max have met: set return value
         rtnVal = min;
      }

      // return the return-value
      return rtnVal;
   }

}
```