DrHabanero: a Platform for Parallel Software Education in Java

Robert “Corky” Cartwright
Vivek Sarkar
Rice University

Programming Pedagogy at Rice

- Functional Programming (FP) is the best starting point
  - simple abstract model of computation directly tied to arithmetic and algebra (see Felleisen et al, “How to Design Programs [HTDP])
  - introduces students to idea of data-directed design using algebraic data
  - formal semantics as laws for simplifying program text
  - test-driven development using templates (e.g. structural recursion) is natural way to make the development process methodical
- OOP is a generalization of FP
  - incremental definition of new abstractions (classes) in terms of existing classes, recursion, and self
  - Composite design pattern provide a straightforward way to encode algebraic types (inductively defined free-term algebras)
  - Command/Strategy pattern is natural encoding of functions as objects
  - Interpreter pattern precisely corresponds the structural recursion template taught in HTDP
  - Our pedagogic IDE (DrJava) supports a functional “subset” of Java focused on programming with immutable algebraic data

Recent Insights

- Trend toward multicore processors is turning programming methodology on its head. Parallel programming [PP] (not concurrent programming) will become dominant modality for developing applications software.
- Most accessible form of PP is functional decomposition into "pure" tasks using "futures"; as in MultiLisp; a "pure" task has a functional specification (maps read-only inputs to an immutable result)
- FP/immutable OOP is easiest way to write "pure" tasks.
- In multicore contexts, much of the copying overhead in future-based FP is dictated by communication and sharing protocols.

Our New Pedagogy

- Functional Programming in Scheme (excerpted from HTDP)
- Immutable OOP in Java (supported by DrJava functional language level)
- Functional subset of Habanero Java [HJ] (supported by DrHabanero)*
- Java enhanced with rich collection of constructs for decomposing computations into parallel tasks; similar in flavor to Cilk but Java rather than C is the base.
- Most tractable subset of HJ is purely functional; tasks are effectively functions from immutable input arguments to immutable results.
- Long term vision: migrate to X10 (which includes analogs for most of our preprocessor constructs and cleans many ugly issues in Java) as it gains mindshare in academic and commerical marketplaces.

Functional HJ

- Functional subset of Java (as DrJava functional level) + 2 constructs
  - **async *expr*** which spawns a future to evaluate ***expr*** asynchronously.
  - **finish *stmt*** which executes ***stmt*** and then blocks until all **async**s spawned (and transitively spawned) in executing ***stmt***
  - **finish** is not essential because the top-level program encloses its body in an implicit **finish**
- The return type of **async** is a **future<*type*>** object which supports the method **get()** , which blocks until the **async** completes (the usual demand operation on a future)
- In Java terms, **async** creates a **Callable** and starts it in an asynchronous thread.
- Possible extensions: array comprehensions, **forall** to excute iterations over a set of points in a region in parallel.

Curriculum Reality

- Start teaching functional programming in Java using DrJava language levels. (Prior exposure to programming but nothing systematic.
- Progress to functional parallel programming followed by mostly functional parallel programming
  - Benign use of imperative code
  - Disciplined use of shared mutable state
- Leverage pedagogic IDEs (DrJava and DrHabanero*)
- Migration to X10 (Scala?) may be our long term salvation.
  - Good notation for functions as data values (Java following suit?)
  - High-level constructs for expressing parallel functional tasks
  - variable/value distinction is critical
  - Successor to Java?

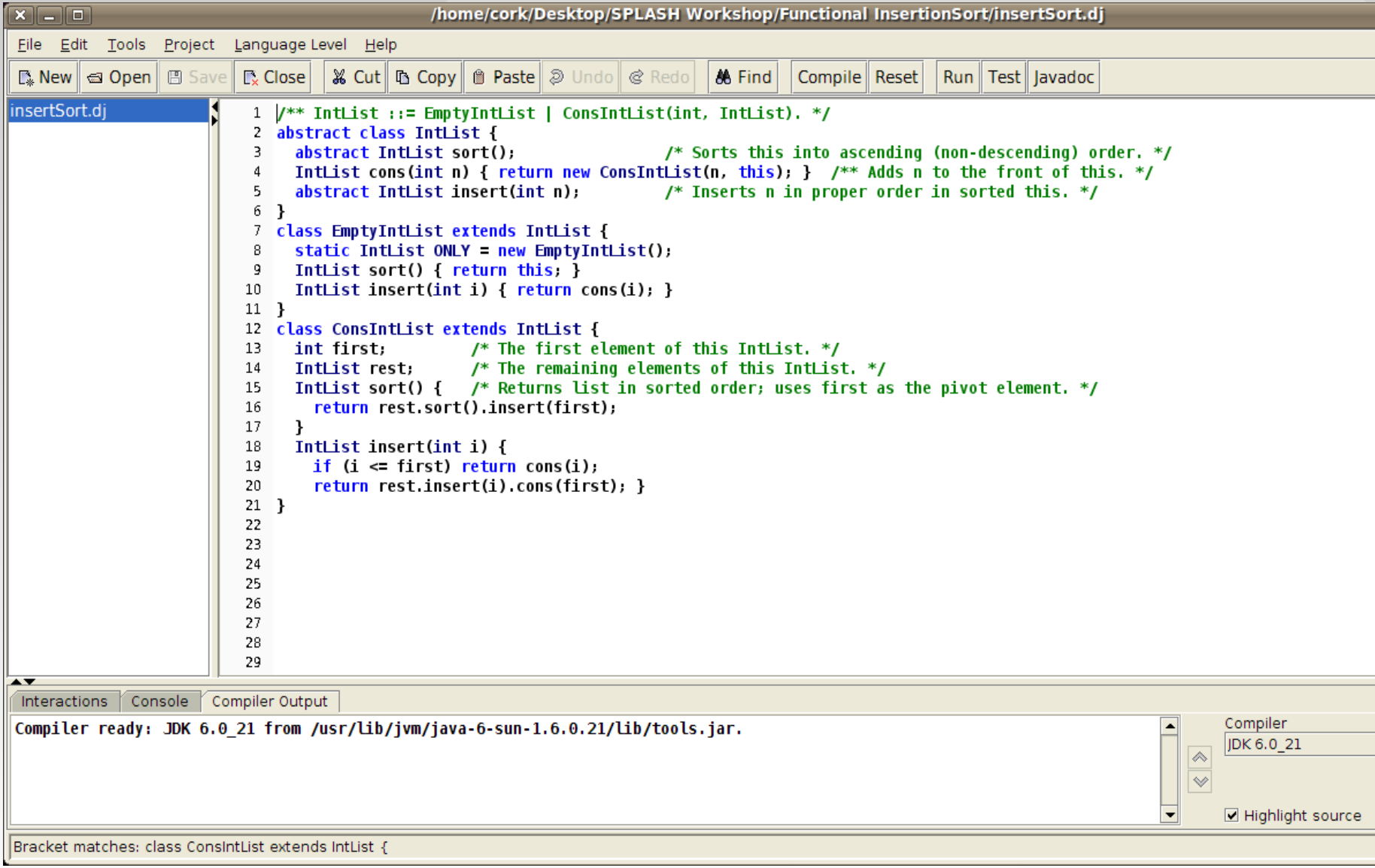
*In development.

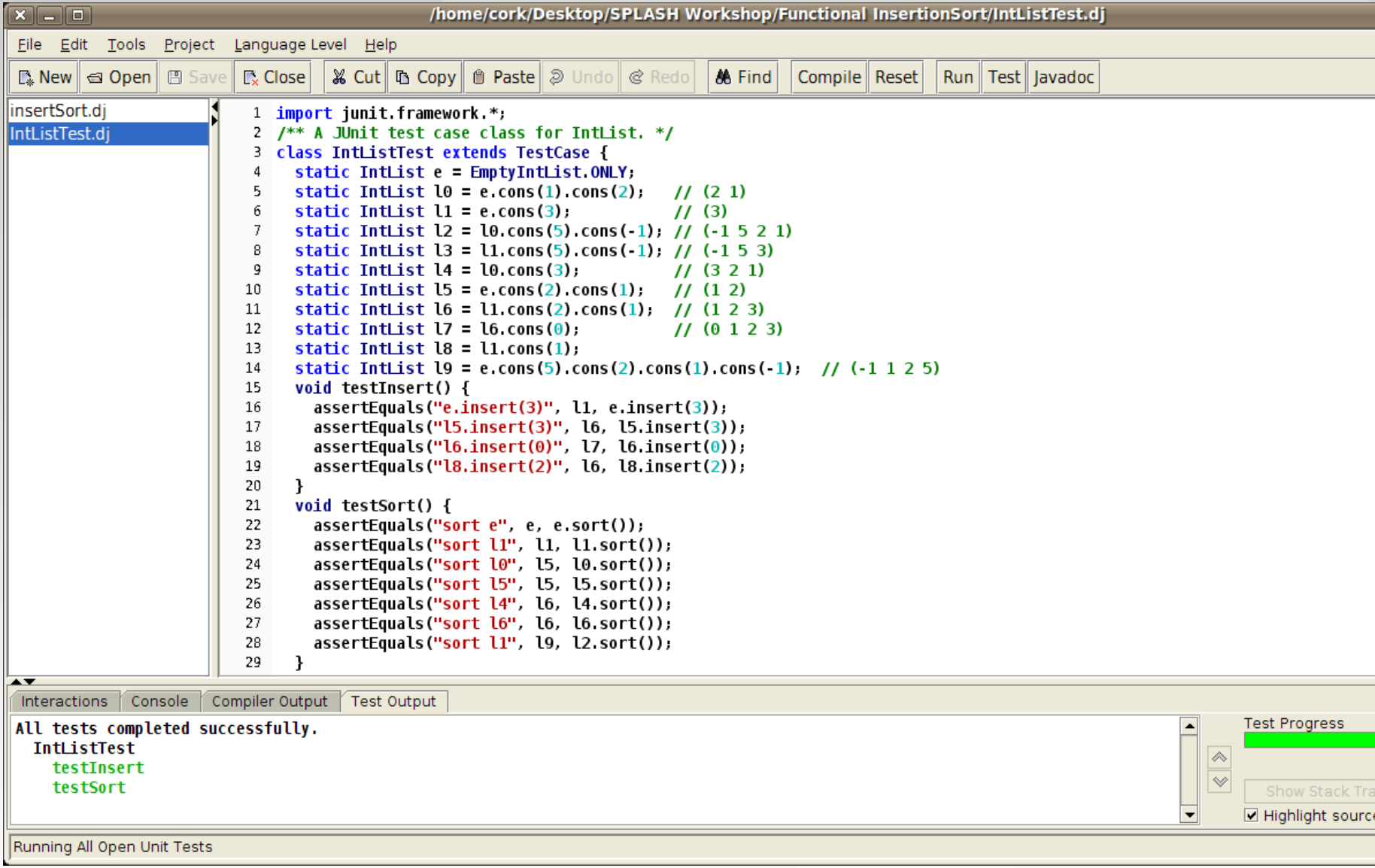
Value of Pedagogic IDEs

- Friendly rather than hostile environment for beginners
  - Syntax highlighting
  - Automatic indentation
- Enforce a particular methodology and associated invariants, *e.g.*, functional Java.
- Provides a framework for supporting new abstractions on top of mainstream language platforms; preprocessing done right.
- Eliminate need for command line execution (and a knowledge of that crufty interface)
- Integrated testing

Demonstration

- DrJava functional language level
  - Insertion sort
  - QuickSort
- DrJava/HJ
  - QuickSort


/home/cork/Desktop/SPLASH Workshop/Functional InsertionSort/insertSort.dj
File Edit Tools Project Language Level Help
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc
insertSort.dj
1 /** IntList ::= EmptyIntList | ConsIntList(int, IntList). */
2 abstract class IntList {
3 abstract IntList sort(); /* Sorts this into ascending (non-descending) order. */
4 IntList cons(int n) { return new ConsIntList(n, this); } /** Adds n to the front of this. */
5 abstract IntList insert(int n); /* Inserts n in proper order in sorted this. */
6 }
7 class EmptyIntList extends IntList {
8 static IntList ONLY = new EmptyIntList();
9 IntList sort() { return this; }
10 IntList insert(int i) { return cons(i); }
11 }
12 class ConsIntList extends IntList {
13 int first; /* The first element of this IntList. */
14 IntList rest; /* The remaining elements of this IntList. */
15 IntList sort() { /* Returns list in sorted order; uses first as the pivot element. */
16 return rest.sort().insert(first);
17 }
18 IntList insert(int i) {
19 if (i <= first) return cons(i);
20 return rest.insert(i).cons(first); }
21 }
22
23
24
25
26
27
28
29
Interactions Console Compiler Output
Compiler ready: JDK 6.0_21 from /usr/lib/jvm/java-6-sun-1.6.0.21/lib/tools.jar.
Compiler
JDK 6.0_21
Highlight source
Bracket matches: class ConsIntList extends IntList {


/home/cork/Desktop/SPLASH Workshop/Functional InsertionSort/IntListTest.dj
File Edit Tools Project Language Level Help
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc
insertSort.dj
IntListTest.dj
1 import junit.framework.*;
2 /** A JUnit test case class for IntList. */
3 class IntListTest extends TestCase {
4 static IntList e = EmptyIntList.ONLY;
5 static IntList l0 = e.cons(1).cons(2); // (2 1)
6 static IntList l1 = e.cons(3); // (3)
7 static IntList l2 = l0.cons(5).cons(-1); // (-1 5 2 1)
8 static IntList l3 = l1.cons(5).cons(-1); // (-1 5 3)
9 static IntList l4 = l0.cons(3); // (3 2 1)
10 static IntList l5 = e.cons(2).cons(1); // (1 2)
11 static IntList l6 = l1.cons(2).cons(1); // (1 2 3)
12 static IntList l7 = l6.cons(0); // (0 1 2 3)
13 static IntList l8 = l1.cons(1);
14 static IntList l9 = e.cons(5).cons(2).cons(1).cons(-1); // (-1 1 2 5)
15 void testInsert() {
16 assertEquals("e.insert(3)", l1, e.insert(3));
17 assertEquals("l5.insert(3)", l6, l5.insert(3));
18 assertEquals("l6.insert(0)", l7, l6.insert(0));
19 assertEquals("l8.insert(2)", l6, l8.insert(2));
20 }
21 void testSort() {
22 assertEquals("sort e", e, e.sort());
23 assertEquals("sort l1", l1, l1.sort());
24 assertEquals("sort l0", l5, l0.sort());
25 assertEquals("sort l5", l5, l5.sort());
26 assertEquals("sort l4", l6, l4.sort());
27 assertEquals("sort l6", l6, l6.sort());
28 assertEquals("sort l1", l9, l2.sort());
29 }
Interactions Console Compiler Output Test Output
All tests completed successfully.
IntListTest
testInsert
testSort
Test Progress
Show Stack Tra
Highlight source
Running All Open Unit Tests

/home/cork/Documents/Rice/Comp 211/Code/Java/IntList/qsort.dj

File Edit Tools Project Language Level Help

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc

qsort.dj
QSortTest.dj

```
1  /** IntList ::= EmptyIntList | ConsIntList(int, IntList). */
2  abstract class IntList {
3    abstract IntList sort();                /* Sorts this into ascending (non-descending) order. */
4    IntList cons(int n) { return new ConsIntList(n, this); }  /** Adds n to the front of this. */
5    abstract IntList append(IntList y);     /* Inserts n in proper order in sorted this. */
6    abstract IntList lessEqual(int pivot);  /* Returns sublist of elements <= pivot. */
7    abstract IntList greater(int pivot);    /* Returns sublist of elements > pivot. */
8  }
9  class EmptyIntList extends IntList {
10   static IntList ONLY = new EmptyIntList();
11   IntList sort() { return this; }
12   IntList append(IntList y) { return y; }
13   IntList lessEqual(int pivot) { return this; }
14   IntList greater(int pivot) { return this; }
15 }
16 class ConsIntList extends IntList {
17   int first;         /* The first element of this IntList. */
18   IntList rest;      /* The remaining elements of this IntList. */
19   IntList sort() {   /* Returns list in sorted order; uses first as the pivot element. */
20     return rest.lessEqual(first).sort(). append(EmptyIntList.ONLY.cons(first)). append(rest.greater(first).sort());
21   }
22   IntList append(IntList y) { return rest.append(y).cons(first); }
23   IntList lessEqual(int pivot) {
24     if (first <= pivot) return rest.lessEqual(pivot).cons(first);
25     return rest.lessEqual(pivot);
26   }
27   IntList greater(int pivot) {
28     if (first <= pivot) return rest.greater(pivot);
29     return rest.greater(pivot).cons(first);
30   }
31 }
32
33
34
35
36
```

Editing /home/cork/Documents/Rice/Comp 211/Code/Java/IntList/qsort.dj 28:51

/home/cork/Desktop/SPLASH Workshop/Functional QuickSort/QSortTest.dj

File Edit Tools Project Language Level Help

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc

qsort.dj
QSortTest.dj

```
1  import junit.framework.*;
2
3  /** A JUnit test case class IntList including the method qsort. */
4  class QSortTest extends TestCase {
5    static IntList e = EmptyIntList.ONLY;
6    static IntList l0 = e.cons(1).cons(2);   // (2 1)
7    static IntList l1 = e.cons(3);           // (3)
8    static IntList l2 = l0.cons(5).cons(-1); // (-1 5 2 1)
9    static IntList l3 = l1.cons(5).cons(-1); // (-1 5 3)
10   static IntList l4 = l0.cons(3);          // (3 2 1)
11   static IntList l5 = e.cons(2).cons(1);   // (1 2)
12   static IntList l6 = l1.cons(2).cons(1);  // (1 2 3)
13   void testAppend() {
14     assertEquals("e.append(e)", e, e.append(e));
15     assertEquals("e.append(l0)", l0, e.append(l0));
16     assertEquals("l0.append(e)", l0, l0.append(e));
17     assertEquals("l1.append(l0)", l4, l1.append(l0));
18   }
19   void testSort() {
20     assertEquals("sort e", e, e.sort());
21     assertEquals("sort l1", l1, l1.sort());
22     assertEquals("sort l0", l5, l0.sort());
23     assertEquals("sort l5", l5, l5.sort());
24     assertEquals("sort l4", l6, l4.sort());
25     assertEquals("sort l6", l6, l6.sort());
26
27     IntList a1 = e.cons(5).cons(2).cons(1).cons(-1);
28     assertEquals("sort l1", a1, l2.sort());
29   }
```

Interactions Console Compiler Output Test Output

```
All tests completed successfully.
  QSortTest
    testAppend
    testSort
```

Test Progress

Show Stack Tra

Highlight source

Running All Open Unit Tests

/home/cork/Desktop/SPLASH Workshop/HJ QuickSort/newQsort.hj

File Edit Tools Project Language Level Help

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc

newQsort.hj

```
1  /** IntList ::= EmptyIntList | ConsIntList(int, IntList). */
2  abstract class IntList {
3    abstract IntList sort();  /** Sorts this into ascending (non-descending) order. */
4    IntList cons(int n) { return new ConsIntList(n, this); }  /** Adds n to the front of this. */
5    abstract IntList append(IntList y);  /** Inserts n in proper order in sorted this. */
6    abstract IntList lessEqual(int pivot);
7    abstract IntList greater(int pivot);
8    public String toString() { return "(" + toStringHelper() + ")"; }
9    abstract String toStringHelper();
10 }
11 class EmptyIntList extends IntList {
12   static IntList ONLY = new EmptyIntList();
13   IntList sort() { return this; }
14   IntList append(IntList y) { return y; }
15   IntList lessEqual(int pivot) { return this; }
16   IntList greater(int pivot) { return this; }
17   public String toString() { return "()"; }
18   String toStringHelper() { return ""; }
19 }
20 class ConsIntList extends IntList {
21   int first;     /* The first element of this IntList. */
22   IntList rest;  /* The remaining elements of this IntList. */
23   ConsIntList(int f, IntList r) { first = f; rest = r; }
24   IntList sort() {
25     final future<IntList> low =  async<IntList> { return rest.lessEqual(first).sort(); };
26     final future<IntList> high =  async<IntList> { return rest.greater(first).sort(); };
27     final IntList mid = EmptyIntList.ONLY.cons(first);
28     return low.get().append(mid).append(high.get());
29   }
30   IntList append(IntList y) { return rest.append(y).cons(first); }
31   IntList lessEqual(int pivot) {
32     if (first <= pivot) return rest.lessEqual(pivot).cons(first);
33     return rest.lessEqual(pivot);
34   }
35   IntList greater(int pivot) {
36     if (first <= pivot) return rest.greater(pivot);
37     return rest.greater(pivot).cons(first);
38   }
39   public String toString() { return "(" + first + rest.toStringHelper() + ")"; }
40   public String toStringHelper() { return " " + first + rest.toStringHelper(); }
41 }
```

Editing /home/cork/Desktop/SPLASH Workshop/HJ QuickSort/newQsort.hj 1:0

/home/cork/Desktop/SPLASH Workshop/HJ QuickSort/qsort.hj

File Edit Tools Project Language Level Help

New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc

qsort.hj

```
1  /** IntList ::= EmptyIntList | ConsIntList(int, IntList). */
2  public abstract class IntList {
3    abstract IntList sort();  /** Sorts this into ascending (non-descending) order. */
4    IntList cons(int n) { return new ConsIntList(n, this); }  /** Adds n to the front of this. */
5    abstract IntList append(IntList y);  /** Inserts n in proper order in sorted this. */
6    abstract IntList lessEqual(int pivot);
7    abstract IntList greater(int pivot);
8    public String toString() { return "(" + toStringHelper() + ")"; }
9    abstract String toStringHelper();
10   public static void main(String[] args) {
11     IntList e = EmptyIntList.ONLY;
12     IntList l0 = e.cons(1).cons(2);   // (2 1)
13     IntList l2 = l0.cons(5).cons(-1); // (-1 5 2 1)
14     System.out.println(l2.sort());
15   }
16 }
17 class EmptyIntList extends IntList {
18   static IntList ONLY = new EmptyIntList();
19   IntList sort() { return this; }
20   IntList append(IntList y) { return y; }
21   IntList lessEqual(int pivot) { return this; }
22   IntList greater(int pivot) { return this; }
23   public String toString() { return "()"; }
24   String toStringHelper() { return ""; }
25 }
26 class ConsIntList extends IntList {
27   int first;     /* The first element of this IntList. */
28   IntList rest;  /* The remaining elements of this IntList. */
29   ConsIntList(int f, IntList r) { first = f; rest = r; }
30   IntList sort() {
31     final future<IntList> low  = async<IntList> { return rest.lessEqual(first).sort(); };
32     final future<IntList> high = async<IntList> { return rest.greater(first).sort(); };
33     final IntList mid = EmptyIntList.ONLY.cons(first);
```

Interactions Console Compiler Output Find/Replace

```
Welcome to DrJava.  Working directory is /home/cork/Desktop/SPLASH Workshop/HJ QuickSort
> run IntList
(-1 1 2 5)
>
```

Running main Method of Current Document