ESC/Java
extended static checking for Java

Erik Poll, Joe Kiniry, David Cok

University of Nijmegen; Eastman Kodak Company
Extended static checker by Rustan Leino et.al. [Compaq].

- *tries to prove correctness of specifications, at compile-time, fully automatically*
Extended static checker by Rustan Leino et.al. [Compaq].

- *tries to prove correctness of specifications, at compile-time, fully automatically*
- *not sound, not complete, but finds lots of potential bugs quickly*
Extended static checker by Rustan Leino et.al. [Compaq].

- *tries to prove correctness of specifications, at compile-time, fully automatically*
- *not sound, not complete, but finds lots of potential bugs quickly*
- good at proving absence of runtime exceptions (eg Null-, ArrayIndexOutOf Bounds-, ClassCast-) and verifying relatively simple properties.
Extended static checker by Rustan Leino et.al. [Compaq].

- *tries to prove correctness of specifications, at compile-time, fully automatically*
- *not sound, not complete, but finds lots of potential bugs quickly*
- good at proving absence of runtime exceptions (eg Null-, ArrayIndexOutOfBounds-, ClassCast-) and verifying relatively simple properties.
- ESC/Java only supported a subset of full JML, but ESC/Java2 by Joe Kiniry [KUN] & David Cok [Kodak] remedies this.
static checking vs runtime checking

Important differences:

- ESC/Java checks specs at compile-time, jmlc checks specs at run-time
- ESC/Java proves correctness of specs, jml only tests correctness of specs.
Hence
  - ESC/Java independent of any test suite, results of runtime testing only as good as the test suite,
  - ESC/Java provided higher degree of confidence.
class Bag {
    int[] a;
    int n;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int mindex = 0;
        for (int i = 1; i <= n; i++) {
            if (a[i] < m) {
                mindex = i; m = a[i];
            }
        }
        n--;
        a[mindex] = a[n];
        return m;
    }
}
class Bag {
    int[] a;
    int n;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int mindex = 0;
        for (int i = 1; i <= n; i++) {
            if (a[i] < m) { mindex = i; m = a[i]; }
        }
        n--; a[mindex] = a[n];
        return m;
    }
}

Warning: possible null dereference. Plus other warnings
class Bag {
    int[] a;  //@ invariant a != null;
    int n;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int minIndex = 0;
        for (int i = 1; i <= n; i++) {
            if (a[i] < m) {
                minIndex = i; m = a[i];
            }
        }
        n--; a[minIndex] = a[n];
        return m;
    }
}
class Bag {
    int[] a;    //@ invariant a != null;
    int    n;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int mindex = 0;
        for (int i = 1; i <= n; i++) {
            if (a[i] < m) {
                mindex = i; m = a[i];
            }
        }
        n--;
        a[mindex] = a[n];
        return m;
    }
}

Warning: Array index possibly too large
class Bag {
    int[] a; //@ invariant a != null;
    int n; //@ invariant 0 <= n && n <= a.length;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int mindex = 0;
        for (int i = 1; i <= n; i++) {
            if (a[i] < m) {
                mindex = i; m = a[i];
            }
        }
        n--;
        a[mindex] = a[n];
        return m;
    }
}
class Bag {
    int[] a;  //@ invariant a != null;
    int n;   //@ invariant 0 <= n && n <= a.length;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int minIndex = 0;
        for (int i = 1; i <= n; i++) {
            if (a[i] < m) {
                minIndex = i; m = a[i];
            }
        }
        n--; a[minIndex] = a[n];
        return m;
    }
}

Warning: Array index possibly too large
class Bag {
    int[] a;  //@ invariant a != null;
    int n;  //@ invariant 0 <= n && n <= a.length;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int minIndex = 0;
        for (int i = 0; i < n; i++) {
            if (a[i] < m) {
                minIndex = i; m = a[i];
            }
        }
        n--;
        a[minIndex] = a[n];
        return m;
    }
}
class Bag {
    int[] a;  // invariant a != null;
    int n;   // invariant 0 <= n && n <= a.length;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int mindex = 0;
        for (int i = 0; i < n; i++) {
            if (a[i] < m) {
                mindex = i; m = a[i];
            }
        }
        n--;
        a[mindex] = a[n];
        return m;
    }
}

Warning: Possible negative array index
class Bag {
    int[] a;  //@ invariant a != null;
    int n;   //@ invariant 0 <= n && n <= a.length;
    //@ requires n > 0;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int mindex = 0;
        for (int i = 0; i < n; i++) {
            if (a[i] < m) {
                mindex = i; m = a[i];
            }
        }
        n--;
        a[mindex] = a[n];
        return m;
    }
}
class Bag {
    int[] a;  // @ invariant a != null;
    int n;   // @ invariant 0 <= n && n <= a.length;
    // @ requires n > 0;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int minIndex = 0;
        for (int i = 0; i < n; i++) {
            if (a[i] < m) {
                minIndex = i;
                m = a[i];
            }
        }
        n--;
        a[minIndex] = a[n];
        return m;
    }
}

No more warnings about this code
class Bag {
    int[] a;     //@ invariant a != null;
    int n;      //@ invariant 0 <= n && n <= a.length;
    //@ requires n > 0;
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int minindex = 0;
        for (int i = 0; i < n; i++) {
            if (a[i] < m) {
                minindex = i; m = a[i];
            }
        }
        n--;
        a[minindex] = a[n];
        return m;
    }
}

...but warnings about calls to extractMin() that do not ensure precondition
Some points to note

- ESC/Java *forces* one to specify some properties.
Some points to note

- ESC/Java *forces* one to specify some properties.

- If you understand the code, then these properties are obvious.

> But for larger programs this may not be the case!
Some points to note

- ESC/Java *forces* one to specify some properties.

- If you understand the code, then these properties are obvious.

  *But for larger programs this may not be the case!*

- If you have these properties documented, then understanding the code is easier.
ESC/Java vs runtime checking (cont.)

- For runtime assertion checking, we could choose what we specify, e.g. all, one, or none of the properties we have written for Bag.

- But for ESC/Java to accept a spec, we are forced to specify all properties (e.g. invariants, preconditions) that this spec relies on.
Limitations of ESC/Java

Like most tools, ESC/Java is

- **not complete**: it may complain about a correct spec
- **not sound**: it may fail to warn about an incorrect spec

ESC/Java warns about many potential bugs, but not about all actual bugs.

These are unavoidable concessions to main goal: pointing out lots of potential bugs quickly & completely automatically

In practice ESC/Java is quite good at checking simple specs, e.g. ruling out any NullPointerException– and IndexOutOfBoundsExceptions