ESC/Java2

extended static checking for Java

Erik Poll

Radboud University Nijmegen

Extended static checker for Java

ESC/Java by Rustan Leino et.al.
Extension ESC/Java2 by David Cok and Joe Kiniry.

- ***tries* to *prove* correctness of JML specifications, at compile-time, fully automatically**

Extended static checker for Java

ESC/Java by Rustan Leino et.al.
Extension ESC/Java2 by David Cok and Joe Kiniry.

- ***tries* to *prove* correctness of JML specifications, at compile-time, fully automatically**
- ***not sound*, *not complete***

  ie. gives some false positives and negatives

  but finds lots of potential bugs quickly

Extended static checker for Java

ESC/Java by Rustan Leino et.al.
Extension ESC/Java2 by David Cok and Joe Kiniry.

- ***tries* to *prove* correctness of JML specifications, at compile-time, fully automatically**
- ***not sound*, *not complete***

  ie. gives some false positives and negatives

  but finds lots of potential bugs quickly
- **good at proving absence of runtime exceptions**

  (eg. Null-, ArrayIndexOutOfBounds-, ClassCast-)

  and verifying relatively simple properties

  (eg. `invariant n > 0`).

ESC/Java2 vs runtime checking

Important differences:

- **ESC/Java2 checks specs at compile-time, jmlrac checks specs at run-time**

ESC/Java2 vs runtime checking

Important differences:

- **ESC/Java2 checks specs at compile-time, jmlrac checks specs at run-time**
- **ESC/Java2 proves correctness of specs, jmlrac only tests correctness of specs:**

ESC/Java2 vs runtime checking

Important differences:

- **ESC/Java2 checks specs at compile-time,**
  jmlrac checks specs at run-time
- **ESC/Java2 proves correctness of specs,**
  jmlrac only tests correctness of specs:
  - **ESC/Java2 independent of any test suite**
    (results of runtime checking only as good as the test suite)

ESC/Java2 vs runtime checking

Important differences:

- **ESC/Java2 checks specs at compile-time,**
  jmlrac checks specs at run-time
- **ESC/Java2 proves correctness of specs,**
  jmlrac only tests correctness of specs:
  - **ESC/Java2 independent of any test suite**
    (results of runtime checking only as good as the test suite)
  - **so it provides higher degree of confidence.**

ESC/Java2 vs runtime checking

Important differences:

- **ESC/Java2 checks specs at compile-time,**
  jmlrac checks specs at run-time
- **ESC/Java2 proves correctness of specs,**
  jmlrac only tests correctness of specs:
  - **ESC/Java2 independent of any test suite**
    (results of runtime checking only as good as the test suite)
  - **so it provides higher degree of confidence.**
  - **but this comes at a price: lots of JML annotations needed . . .**

ESC/Java2 “demo”

```
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;
 }
```

ESC/Java2 “demo”

```
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 deference. Plus other warnings

ESC/Java2 “demo”

```
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;
 }
```

ESC/Java2 “demo”

```
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

ESC/Java2 “demo”

```
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;
 }
```

ESC/Java2 “demo”

```
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;
 }
```

Warning: Array index possibly too large

ESC/Java2 “demo”

```
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;
 }
```

ESC/Java2 “demo”

```
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

ESC/Java2 “demo”

```
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;
 }
```

ESC/Java2 “demo”

```
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;
  }
```

No more warnings about this code

ESC/Java2 “demo”

```
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;
```

...but warnings about calls to `extractMin()` that do not ensure precondition

Some points to note

- **ESC/Java2 *forces* to specify some properties.**

Some points to note

- **ESC/Java2 *forces* 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/Java2 *forces* 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/Java2 vs runtime checking

- For runtime assertion checking, we could choose what we specify, e.g. all, one, or none of the properties we have written for for Bag.
- But for ESC/Java2 to accept a spec, we are *forced* to specify *all properties* that this spec relies on.

ESC/Java2 vs runtime checking

Runtime assertion checking

- low cost & effort
- easy to do as part of normal testing

ESC/Java2 vs runtime checking

Runtime assertion checking

- low cost & effort
- easy to do as part of normal testing

Extended static checking with ESC/Java2

- higher cost & effort
- possible for JavaCard-sized programs
- higher assurance: independent of any test suite
- checking a spec with ESC/Java *forces* you to specify all the invariants and API contracts that it relies on

Limitations of ESC/Java2

- **ESC/Java2 is**
  - **not sound: may warn about correct spec**
  - **not complete: may fail to warn about incorrect spec**

  These are unavoidable concessions to main goal: finding lots of potential bugs, completely automatically
  In practice neither issue is much of a problem.
- **Using ESC/Java2 is only really feasible on small programs, because it's so labour-intensive.**

Using ESC/Java2 in this course

- **ESC/Java2 can cope with JavaCard sized programs**
- **Later you will have to use ESC/Java2 on your smartcard applet, to prove that it will never thrown unexpected exceptions and possible additional properties that you can think of**
- **This week: small home-work assignment using ESC/Java2.**