Runtime vs. Static Checking

Runtime Checking
- finds bugs at run-time,
- tests for violation during execution,
- can check most of the JML,
- is done by jmlrac.

Static Checking
- finds bugs at compile-time,
- proves that there is no violation,
- can check only parts of the JML,
- is done by ESC/Java.
ESC/Java and run-time exceptions

ESC/Java checks that no undeclared run-time exceptions occur.

- NullPointerException
- ClassCastException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- ArithmeticException
- NegativeArraySizeException
- other run-time exception, e.g., when calling library functions.
ESC/Java and specification

ESC/Java also checks the JML specification:
- **ensures** in method contract,
- **requires** in called methods,
- **assert** statements,
- **signals** clause,
- **invariant** (loop invariant and class invariant).

ESC/Java assumes that some formulae hold:
- **requires** in method contract,
- **ensures** in called methods,
- **assume** statements,
- **invariant** (loop invariant and class invariant).
public void put(Object o) {
    int hash = o.hashCode();
    ...
}

results in Possible null dereference.

Solutions:

- Declare o as non_null.
- Add o != null to precondition.
- Add throws NullPointerException.
  (Also add signals (NullPointerException) o == null)
- Add Java code that handles null pointers.
  \[
  \text{int hash} = (o == null ? 0 : o.hashCode());
  \]
class Priority implements Comparable {
    public int compareTo(Object other) {
        Priority o = (Priority) other;
        ...
    }
}

results in Possible type cast error.
Solutions:

- Add throws ClassCastException.
  (Also add signals (ClassCastException) !(other instanceof Priority))

- Add Java code that handles differently typed objects:
  if (!(other instanceof Priority))
      return -other.compareTo(this)
  Priority o = ...

This results in a Possible null dereference.
ArrayIndexOutOfBoundsException

```java
void write(/*@non_null@*/ byte[] what, int offset, int len) {
    for (int i = 0; i < len; i++) {
        write(what[offset+i]);
    }
}
```

results in Possible negative array index

Solution:

- Add `offset >= 0` to pre-condition, this results in Array index possibly too large.
- Add `offset + len <= what.length`.
- ESC/Java does not complain but there is still a problem. If `offset` and `len` are very large numbers, then `offset + len` can be negative. The code would throw an ArrayIndexOutOfBoundsException at run-time.
- The correct pre-condition is:
  ```java
  /*@ requires offset >= 0 && offset + len >= offset
   @     && offset + len <= what.length;
  @*/
  ```
public class Stack {
    /*@non_null@*/ Object[] elems;
    int top;
    /*@invariant 0 <= top && top <= elems.length @*/

    /*@ requires top < elems.length; @*/
    void add(Object o) {
        elems[top++] = o;
    }
}

results in Type of right-hand side possibly not a subtype of array element type (ArrayStore).

Solutions:

- Add an invariant $\text{typeof}(\text{elems}) == \text{typeof}(\text{Object[]})$.
- Add a precondition $\text{typeof}(o) <: \text{elemtype}(\text{typeof}(\text{elems}))$. 
Types in assertions

- `typeof` gets the run-time type of an expression.
  \[
  \texttt{typeof(obj)} \sim \texttt{obj.getClass()}.  
  \]

- `elemtype` gets the base type from an array type.
  \[
  \texttt{elemtype(t1)} \sim \texttt{t1.getComponentType()}.  
  \]

- `type` gets the type representing the given Java type.
  \[
  \texttt{type(Foo)} \sim \texttt{Foo.class}  
  \]

- `<:` means is sub-type of.
  \[
  t1 <: t2 \sim t2.isAssignableFrom(t1)  
  \]
class HashTable {
    /*@non_null@*/ Bucket[] buckets;
    void put(/*@non_null@*/ Object key, Object val) {
        int hash = key.hashCode() % buckets.length;
        ...
    }
}

results in Possible division by zero.
Solution:

- Add invariant buckets.length > 0.
- Run ESC/Java again to check that this invariant holds.
- It probably warns about a Possible negative array index.
class Bag {
    /*@ non_null @*/ Object[] elems;

    void sort() {
        java.util.Arrays.sort(elems);
    }
}

results in Possible unexpected exception.

- Look in escjava/specs/java/util/Arrays.refines-spec!
- Array.sort() has pre-condition:
  
  elems[i] instanceof Comparable for all i.
- Solution: Add similar condition as class invariant.
Assume and Assert

The basic specifications in ESC/Java are **assume** and **assert**.

```java
/*@ assume this.next != null; @*/
this.next.prev = this;
/*@ assert this.next.prev == this; @*/
```

- ESCJava proves that if the assumption holds in the pre-state, the assertion holds in the post-state.
- This is a **Hoare triple**.
The method specification is just translated into **assume** and **assert**:

```java
/*@ requires n > 0; */
@ ensures \result \text{==} (\text{int}) Math.sqrt(n);@*/
int m() {
    ...
    return x;
}
```

is treated as:

```java
/*@ assume n > 0; endwhile */
...
/*@ assert x \text{==} (\text{int}) Math.sqrt(n); endwhile */
```
And if $m()$ is called the assumption and assertion is the other way round:

```java
... 
  y = m(x);
  ...

is treated as

```java
... 
  /**< assert x > 0; */
  y = m(x);
  /**< assume y == (int) Math.sqrt(x); */
  ...
```
Checking for Exceptions

To check for run-time exceptions ESC/Java automatically inserts asserts:

```java
a[x] = "Hello";
```

is treated as:

```java
/*@ assert a != null && x >= 0 && x < a.length &&
   typeof("Hello") <: elemtype(typeof(a)); */
a[x] = "Hello";
```
Never assume something wrong. This enables ESC/Java to prove everything:

```java
Object o = null;
/*@ assume o != null; @*/
Object[] a = new String[-5];
a[-3] = new Integer(2);
```

> escjava2 -q AssumeFalseTest.java
0 warnings
ESC/Java can only do limited reasoning:

```java
/*@ requires i == 5 && j == 3;
  @ ensures result == 15;
  @*/

int m(int i, int j) {
  return i * j;
}
```

Test.java:19: Warning: Postcondition possibly not established (Post)

```
^}
```

Associated declaration is "Test.java", line 14, col 8:

```java
@ ensures result == 15;
```

A good assumption can help, e.g.

```java
int m(int i, int j) {
  /*@ assume 15 == 5 * 3; @*/
  return i * j;
}
```

But this may introduce unsoundness if not used carefully.
Loops in ESC/Java

```java
int a[] = new int[6];
for (int i = 0; i <= 6; i++) {
    a[i] = i;
}
```

> escjava2 -q Test.java
0 warnings

> escjava2 -Loop 7 -q Test.java
Test.java:15: Warning: Array index possibly too large (IndexTooBig)
    a[i] = i;
   ^
1 warning

> escjava2 -LoopSafe -q Test.java
Test.java:15: Warning: Array index possibly too large (IndexTooBig)
    a[i] = i;
   ^
1 warning
Demo