Simple Example

- We'll go through the Bag.java example from the manual
- Illustrates
  - basic checks performed on unannotated code
  - Addition of assumptions
  - Addition of preconditions
```java
class Bag {
    int[] a;
    int n;

    Bag(int[] input) {
        n = input.length;
        a = new int[n];
        System.arraycopy(input, 0, a, 0, 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;
    }
}
```
Running ESC/Java

prompt>escjava -quiet Bag.java
Bag.java:6: Warning: Possible null dereference (Null)
    n = input.length;
   ^
Bag.java:15: Warning: Possible null dereference (Null)
    if (a[i] < n) {
   ^
Execution trace information:
    Reached top of loop after 0 iterations in
    "Bag.java", line 14, col 4.

2 warnings

class Bag {
    int[] a; Null?
    int n;

    Bag(int[] input) {
        n = input.length;
        a = new int[n];
        System.arraycopy(input, 0, a, 0, n);
    }
}
class Bag {
    int[] a;
    int n;

    //@ requires input != null;
    Bag(int[] input) {
        n = input.length;
        a = new int[n];
        System.arraycopy(input, 0, a, 0, n);
    }
}

Running ESC/Java

prompt>escjava -quiet Bag.java
Bag.java:15: Warning: Possible null dereference (Null)
    if (a[i] < m) {
        ^
Execution trace information:
    Reached top of loop after 0 iterations

2 warnings
```java
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;
}
```

**Modular Checking**

- Analysis of `extractMin` doesn’t know that the constructor always ensures that `a` is not null
- It also doesn’t know whether the constructor will be called before `extractMin` (although it could)
- Specify this property of `a`
```java
class Bag {
    /*@ non_null */ int[] a;
    int n;

    //@ requires input != null;
    Bag(int[] input) {
        n = input.length;
        a = new int[n];
        System.arraycopy(input, 0, a, 0, n);
    }
}
```

Running ESC/Java

```
prompt>escj ava -quiet Bag.java
Bag.java:16: Warning: Array index possibly too large (IndexTooBig)
    if (a[i] < n) {
          ^
Execution trace information:
    Reached top of loop after 0 iterations in "Bag.java", line 15, col 4.
2 warnings
```
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;
}
}
Running ESC/Java

prompt > escj ava -qui et Bag.java
Bag.java:16: Warning: Array index possibly too large (IndexTooBig)
    if (a[i] < n) {
    ^
Execution trace information:
   Reached top of loop after 0 iterations in "Bag.java", line 15, col 4.

2 warnings

Different from manual

Problem is Modularity

- Analysis of extractMin doesn’t see that all a fields have bounds of n and n only decreases after the allocation
- Several options
  - Local assumption, precondition
  - Global property of field so we add invariant 0<=n && n<=a.length;
class Bag {
    //@ non_null */ int[] a;
    int n;
    // @invariant 0 <= n && n <= a.length;

    //@ requires input != null;
    Bag(int[] input) {
        n = input.length;
        a = new int[n];
        System.arraycopy(input, 0, a, 0, n);
    }
}

Running ESC/Java

prompt>escjava -quiet Bag.java
Bag.java:23: Warning: Possible negative array index (IndexNegative)
    a[mindex] = a[n]
^

Execution trace information:
    Reached top of loop after 0 iterations

2 warnings
What about the Invariant?

- It constrains the value of \( n \) to be non-negative.
- As in JML, invariants are enforced at method boundaries (call and return).
- If we call this method with \( n = 0 \) that satisfies the invariant:
  - but it will result in the index exception.
Possible Solutions

- Add a special case to avoid decrementing $n$ below 0
- Require that $n$ be positive on entry to the `extractMin` method
- We’ll try both

```java
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];
        }
    }
    if (n > 0) {
        n--;
        a[mindex] = a[n];
    }
    return m;
}
```
// @requires n>=1;
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;
}

Running ESC/Java

prompt>escj ava Bag. java
ESC/Java version 1.2.2, 12 October 2000

Bag . . .

Bag: Bag(int[]) . . .
    [0.701 s] passed

Bag: extractMin() . . .
    [0.2 s] passed
    [3.435 s total]
For You to Do

- What additional specifications can you think of writing?
- Add them and try to check them

Catalog of Errors

- ESC Java reports a variety of errors
- Built-in JVM checks
  - Cast, Deadlock, NegSize, IndexNegative, IndexTooBig, ArrayStore, Null, Uninit, ZeroDiv
- Explicit annotation violations
  - Assert, Invariant, LoopInv,NonNull, NonNullInit, Pre, Post, Reachable
  - Exception [throw not in exsures]
Uniqueness Constraints

- When ESC/Java analyzes a method body, it assumes that other Java statements may be executing simultaneously.
- This can lead to some surprises for users who take a purely sequential viewpoint.
- Options
  - Assume there is no concurrency (very unsafe)
  - Analyze program for interference
  - Let programmers specify non-interference related properties

Ownership

- If an object, o, has a reference field that points to object, p, and p’s reference is not stored elsewhere in the program, then we say that o “owns” p.
- This is a special form of a uniqueness constraint.
- ESC/Java supports this via the
  - Implicit “owner” attribute of each object
  - “set” command to establish ownership
Consider the following example

```java
class ObjStack {
    Object[] a;
    int n;
    //@ invariant (\forall int i; n <= i & i < a.length ==> a[i] == null);

    ObjStack(int l) {
        n = 0;
        a = new Object[l];
    }

    void Push(Object o) {
        a[n++] = o;
    }

    Object Pop() {
        Object o = a[--n];
        a[n] = null;
        return o;
    }
}
```

escjava -suggest

ESC/Java version 1.2.4, 27 September 2001

ObjStack ...

ObjStack: ObjStack(int) ...

objstack.java:12: Warning: Possible attempt to allocate array of negative length (NegSize)
    a = new Object[l];
  ^

Suggestion [12,20]: perhaps declare constructor 'ObjStack' at 10,4 in objstack.java with 'requires 0 <= l;'

[0.36 s] failed

...
Consider the following example

class ObjStack {
    /*! @ non_null */ Object [] a;
    //@ invariant \elemtype(typeof(a)) == \type(Object);
    int n;
    //@ invariant (\forall int i; n <= i & i < a.length ==> a[i] == null);

    // requires l > 0;
    ObjStack() {
        n = 0;
        a = new Object[l];
    }

    void Push(Object o) {
        a[n++] = o;
    }

    Object Pop() {
        Object o = a[--n];
        a[n] = null;
        return o;
    }
}

escjava -suggest

ObjStack: Pop() ...
------------------------------------------------------------------------
objstack.java:22: Warning: Possible negative array index (IndexNegative)
    Object o = a[--n];
      ^
Suggestion [22,18]: none <big expression>
------------------------------------------------------------------------

In this case we can look to the arguments of the methods and define checkable annotations constraining n’s range
Consider the following example

class ObjStack {
    /*@ non_null */ Object [] a;
    //@ invariant @elemtype(a) == type(Object);
    int n;  //@ invariant 0 <= n & n <= a.length;
    //@ invariant (forall int i; n <= i & i < a.length ==> a[i] == null);
    //@ requires l > 0;

    ObjStack() {
        n = 0;
        a = new Object[l];
    }

    //@ requires n < a.length;
    void Push(Object o) {
        a[n++] = o;
    }

    //@ requires n > 0;
    Object Pop() {
        Object o = a[--n];
        a[n] = null;
        return o;
    }
}

escjava -suggest

ObjStack: Pop() ...
ObjStack: Push(java.lang.Object) ...
------------------------------------------------------------------------
objstack.java:18: Warning: Possible violation of object invariant (Invariant)
^ 
Associated declaration is "objstack.java", line 6, col 8:
/*@ invariant (forall int i; n <= i & i < a.length ==> a[i] == null);
^ 
Possibly relevant items from the counterexample context:
(0 <= (brokenObj<2>).n:17.8)
((brokenObj<2>).n:17.8 <= i:6.31)
(isAllocated(brokenObj<2>) < alloc)
(typeof(brokenObj<2>) <: T_java.lang.Object)
(brokenObj<2>).a:17.6 == tmp0:3.30
(typeof(brokenObj<2>) <: T_ObjStack)
classDown(T_java.lang.Object)(typeof(brokenObj<2>)) == T_ObjStack
(brokenObj<2>).n:17.8 == (brokenObj<2>).n:17.8
(brokenObj<2>) == this
(brokenObj<2> != null)
(brokenObj<2> refers to the object for which the invariant is broken.)
Suggestion [18,4]: perhaps declare instance invariant 'invariant this.a.owner==
this;' in class ObjStack (near associated declaration at "objstack.java", line 6, col 8)
------------------------------------------------------------------------
[0.24 s] failed
What’s happening here

- ESC/Java doesn’t know that instances of ObjStack “own” their array’s (a)
- It assumes there are two ObjStacks

Consider the following example

```java
//@ invariant this.a.owner == this;
class ObjStack {  
    /*@ non_null */ Object [] a;  
    //@ invariant @elmentype(typeof(a)) == @type(Object);  
    int n;  //@ invariant 0 <= n & n <= a.length;  
    //@ invariant (forall int i; n <= i & i < a.length ==> a[i] == null);  
    // requires l > 0;  
    ObjStack() {  
        n = 0;  
        a = new Object[l];  
        //@ set a.owner = this;  
    }
    //@ requires n < a.length;  
    void Push(Object o) {  
        a[n++] = o;  
    }
    //@ requires n > 0;  
    Object Pop() {  
        Object o = a[--n];  
        a[n] = null;  
        return o;  
    }
}
```
**escjava -suggest**

ESC/Java version 1.2.4, 27 September 2001

ObjStack ...

ObjStack: ObjStack(int) ...
  [0.301 s] passed

ObjStack: Push(java.lang.Object) ...
  [0.1 s] passed

ObjStack: Pop() ...
  [0.11 s] passed
  [0.961 s total]

---

**Interface Specifications**

- Break visibility for purpose of spec
  - `spec_public` f [private field f]
- Makes f visible at interface’
- Workaround for lack of calls in exprs
- Introduce specification only data
  - `ghost` [declare model field]
  - `set` [assign to model field]
- Example: Interface spec of queue
public class Queue {
    //@ ghost public int size;
    //@ invariant size >= 0;
    //@ ghost public \TYPE elementType;
    //@ ghost public boolean canHoldNull;

    //@ ensures elementType == \type/Object();
    //@ ensures canHoldDNull;
    //@ ensures size == 0;
    public Queue();

    • elementType records base type for all queue elements
    • canHoldNull determines whether nulls can be inserted

    bindActionCreators
    /*@ requires \typeof(e) <: elementType | (e == null & canHoldDNull); */
    //@ modifies size;
    //@ ensures size == \old(size) + 1;
    public void enqueue(Object e);

    /*@ ensures \result == (size == 0); */
    public boolean isEmpty();

    /*@ ensures size >= 1; */
    /*@ modifies size; */
    /*@ ensures \typeof(\result) <: elementType | (\result == null & canHoldDNull); */
    //@ ensures size == \old(size) - 1;
    public Object dequeue();
}
Subtypes of Queue

- Are obliged to satisfy pre/post conditions mentioned of supertype
- The representation of specification fields must be encoded explicitly in subtypes
  - Insert `set` pragmas for appropriate ghost variables at appropriate points in the implementation
    ```
    //@ set size = size+1;
    theVector.addElement(o);
    ```

Clients of Queue

- Can enable checking of pre conditions and assumption of post-conditions by setting ghost variables
  ```
  Queue q = new SomeQueue();
  //@ set q.elementType = \type(T);
  //@ set q.canHoldNull = false;
  ```
For You to Do

- Take your favorite implementation of a queue and implement it as a subtype of this interface
- Figure out how to map your fields onto the ghost fields
- Set elementType and canHoldNull
- Check to see if your implementation satisfies the spec