Whose fault is it when a precondition fails?

- The implementer of the class: 3
- The client: 16
- Bertrand Meyer: 3
Whose fault is it when an invariant fails?

- The implementer of the class: 16
- The client: 0
- James Gosling: 2
What variables can I reference in a precondition?

- My own state (instance variables): 18
- My inherited state: 12
- Arguments to the method: 13
- The result of the method: 2
- My state at the end of the method: 3
- State of my client: 6
- State of my collaborators: 4
- Global variables: 9
What variables can I reference in a postcondition?

- My own state (instance variables): 20
- My inherited state: 10
- Arguments to the method: 6
- The result of the method: 18
- My state at the end of the method: 23
- State of my client: 3
- State of my collaborators: 6
- Global variables: 3
### What variables can I reference in a class invariant?

<table>
<thead>
<tr>
<th>Variable</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>My own state (instance variables)</td>
<td>20</td>
</tr>
<tr>
<td>My inherited state</td>
<td>18</td>
</tr>
<tr>
<td>Arguments to the constructor</td>
<td>12</td>
</tr>
<tr>
<td>The result of the constructor</td>
<td>15</td>
</tr>
<tr>
<td>My state at the end of the constructor</td>
<td>13</td>
</tr>
<tr>
<td>State of my client</td>
<td>3</td>
</tr>
<tr>
<td>State of my collaborators</td>
<td>2</td>
</tr>
<tr>
<td>Global variables</td>
<td>3</td>
</tr>
</tbody>
</table>
How can you get rid of null values in your code?

- be careful
- initialize properly
- insert nil instead of null (like in SQL), make intern checks
- make a Null object
- Don't return null
- use a specific object as a stand in for null values
- try{}catch
- use null objects to return instead of returning null values
- Use a null object
How can you get rid of null values in your code?

- Maybe you could make a class that allows "null-objects"
- assert something is not null, before returning
- Use a special object for null

So that we can access the invariant from a child class
What would be a full class invariant for LinkStack?

- size overflow
- when the return is -1?
- Call from methods outside this class
Why did we declare invariant as protected and not private?

- So we can call it from a child class
- so that subclasses can override the invariant?
- when you have to look at them in other methods/class
- Useful for subclasses
- so that a "subclass stack" can use it
Why did ParenMatch fail?

- The stack was not empty
- cause after the fail still a parenthese is left on the stack and that's why the next one that should be valid fails
- It did not reset the stack after the first fail

```
public class ParenMatch {
  "public class ParenMatch {" is not balanced
  factorial(10);
  "factorial(10);" is not balanced
}
"}" is balanced
```
What are the missing contracts in the ParenMatch class?

- check that the stack is empty as a precondition
Last chance for questions