Null Check Analysis in Java Code

Manuel Leuenberger

Software Composition Seminar

05.05.2015
Motivation

• NullPointerException-related bugs are the most frequent in Java projects

• Often fixed by guarding problematic code with a null check
• What are the *things* that are checked against null?
• Where are these *things* coming from?
public class CoDriver {
    ...
    public void join(Driver driver) {
        ...
        Car car;
        ...
        car = driver.getCar();
        ...
        if (car != null) {
            ...
        }
    }
}
public class CoDriver {
  
  public void join(Driver driver) {
    Car car;

    if (car != null) {
      
    }
  }
}
public class CoDriver {
    ...
    public void join(Driver driver) {
        ...
        Car car;
        ...
        car = driver.getCar();
        ...
        if (car != null) {
            ...
        }
    }
}
public class CoDriver {
    ... 
    public void join(Driver driver) {
        ... 
        Car car;
        ... 
        car = driver.getCar();
        ... 
        if (car != null) {
            ... 
        }
    }
}
public class CoDriver {
    ...
    public void join(Driver driver) {
        ...
        Car car;
        ...
        car = driver.getCar();
        ...
        if (car != null) {
            ...
        }
    }
}
public class CoDriver {
    ... 
    public void join(Driver driver) {
        ... 
        Car car;
        ... 
        car = driver.getCar();
        ... 
        if (car != null) {
            ... 
        }
    }
}
public class CoDriver {
    ...  
    public void join(Driver driver) {
        ...  
        Car car;  
        ...  
        car = driver.getCar();  
        ...  
        if (car != null) {
            ...  
        }
    }
}
public class CoDriver {
    ...
    public void join(Driver driver) {
        ...
        Car car; // local variable
        ...
        car = driver.getCar();
        ...
        if (car != null) {
            ...
        }
    }
}
public class CoDriver {
    ...
    public void join(Driver driver) {
        ...
        Car car;
        ...
        car = driver.getCar();
        ...
        if (car != null) {
            ...
        }
    }
}
public class CoDriver {
    ...
    public void join(Driver driver) {
        ...
        Car car;
        ...
        car = driver.getCar();
        ...
        if (car != null) {
            ...
        }
    }
}
public class CoDriver {
    ...
    public void join(Driver driver) {
        ...
        Car car;
        ...
        car = driver.getCar();
        ...
        if (car != null) {
            ...
        }
    }
}
public class CoDriver {
    ...
    public void join(Driver driver) {
        ...
        Car car;
        ...
        car = driver.getCar();
        ...
        if (car != null) {
            ...
        }
    }
}
public class CoDriver {
    ...
    public void join(Driver driver) {
        ...
        Car car;
        car = driver.getCar();
        ...
        if (car != null) {
            ...
        }
    }
}
Dataset

- 717 projects
- 374’203 Java sources processed
- 6’106 Java sources unprocessable
35% of conditionals are null checks!
Comparand are...

![Graph showing the percentage of occurrences of different comparand types]

- **Name**: 78%
- **MethodCall**: 15%
- **FieldAccess**: 5%
- **Enclosed**: 1%
- **ArrayAccess**: 1%
- **Binary**: 0%
- **Cast**: 0%
- **StringLiteral**: 0%
- **NullLiteral**: 0%
- **Class**: 0%
- **ObjectCreation**: 0%
- **IntegerLiteral**: 0%
Assignable comparands are...

<table>
<thead>
<tr>
<th>Type</th>
<th>% of #Occurences</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalVariable</td>
<td>50%</td>
</tr>
<tr>
<td>MemberVariable</td>
<td>24%</td>
</tr>
<tr>
<td>Parameter</td>
<td>23%</td>
</tr>
<tr>
<td>Undefined</td>
<td>3%</td>
</tr>
</tbody>
</table>
Assignable comparands come from…

- MethodCall: 71%
- NullLiteral: 10%
- Cast: 7%
- ObjectCreation: 4%
- Name: 3%
- Conditional: 2%
- FieldAccess: 1%
- ArrayAccess: 1%
- StringLiteral: 0%
- Binary: 0%
- Enclosed: 0%
- ArrayCreation: 0%
- This: 0%
- Assign: 0%
- Class: 0%
- IntegerLiteral: 0%
- BooleanLiteral: 0%
- Unary: 0%
- LongLiteral: 0%
- DoubleLiteral: 0%
- ArrayInitializer: 0%
- CharLiteral: 0%
Conclusion

- We check for null, because methods return null
- …but we often mistakenly do not expect null
Lessons Learned

- Spring Batch: Too big to handle
- Performance: Caching is your friend
- MySQL: Question your very basic mental model
+ Java 8 Streams: Process collections like a boss
+ GraphViz: Got graph data? Got visualization!