Nullable How-To

Intro

This how-to document describes `@Nullable` and `@NotNull` annotations introduced in IntelliJ IDEA for catching `NullPointerException's` (NPE's) through the **Constant Conditions & Exceptions** and **@Nullable problem** inspections.

Code Inspection is an IntelliJ IDEA tool to maintain and clean up your code allowing you to find bugs or inconsistencies and suggesting automated solutions if possible. **Constant Conditions & Exceptions** is an inspection detecting:

- single-state conditions (expressions that are always `true` or `false`)
- method invocations and field dereferences that might throw the NPE

The `@Nullable` and `@NotNull` annotations are designed to help you watching contracts throughout method hierarchy to avoid emergence of NPE's. Moreover, IntelliJ IDEA provides another benefit for them: the **Code Inspection** mechanism informs you on such contracts' discrepancies in places where annotated methods are called and provides automated solutions in some cases.

These annotations are proprietary ones and included in the bundled JAR. We at JetBrains suggested to include these annotations in the standard Java SDK. The issue is still pending.

Currently the annotations are distributed under the Apache license. The source code is supplied as well.

@Nullable and @NotNull - What for and How to Start Using?

Two annotations – `@Nullable` and `@NotNull` – handle method invocations and field dereferences outside methods.

The `@Nullable Annotation` reminds you on necessity to introduce an NPE check when:

- calling methods that can return `null`
- dereferencing variables (fields, local variables, parameters) that can be `null`

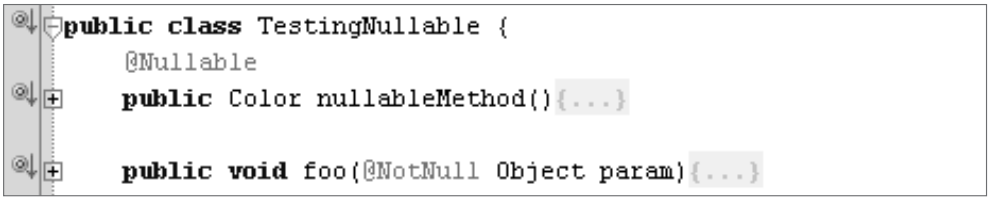
The `@NotNull Annotation` is, actually, an explicit contrast declaring the following:

- a method should not return `null`
- a variable (like fields, local variables, and parameters) cannot hold the `null` value

IntelliJ IDEA warns you if these contracts are violated.

To use the `@Nullable` and `@NotNull` annotations:

1. Add (for instance, to your module or project) a library called `annotations.jar`. It can be found in the <IntelliJ_IDEA_Home>/redist folder.
2. Then the desired annotation is to be introduced before the method or variable declaration.



@Nullable and @NotNull - Formal Semantics

An element annotated with `@Nullable` claims the `null` value is perfectly *valid* to return (for methods), pass to (for parameters) and hold (for local variables and fields).

An element annotated with `@NotNull` claims the `null` value is *forbidden* to return (for methods), pass to (for parameters) and hold (for local variables and fields).

There is a covariance-contravariance relationship between `@Nullable` and `@NotNull` when overriding/implementing methods with annotated declaration or parameters.

- Overriding/implementing methods with an annotated declaration:
  - The `@NotNull` annotation of the parent method requires the `@NotNull` annotation for the child class method.
  - Methods with the `@Nullable` annotation in the parent method can have either `@Nullable` or `@NotNull` annotations in the child class method.
- Overriding/implementing methods with annotated parameters:
  - The `@Nullable` annotation of the parameter in the parent method requires the `@Nullable` annotation for the child class method parameter.
  - Methods with the `@NotNull` annotation of the parameter in the parent method can have either `@Nullable` or `@NotNull` annotations (or none of them) for the child class method parameter.

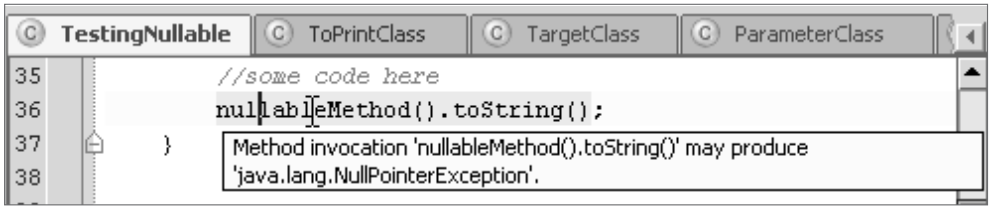
@Nullable Annotation - Examples

The `@Nullable` annotation helps you to find method invocations that potentially might return `null`. In such case, you can make IntelliJ IDEA to warn you explicitly that using such method's results you must check them for being `null`.

For instance, let's take the following code:

```
public class TestingNullable {
   @Nullable
   public Color nullableMethod(){
      //some code here

      return color;
   }

   public void usingNullableMethod(){
      // some code

      nullableMethod().toString();

      // Here using the nullableMethod() method's result
      // without checking for null can produce an NPE.
   }
}
```

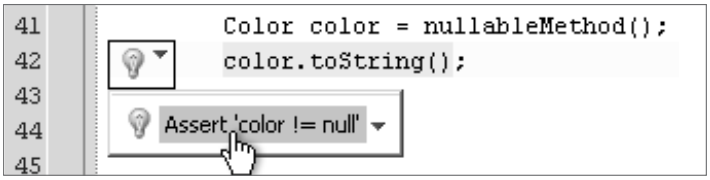
And this is how it looks in the IntelliJ IDEA UI.



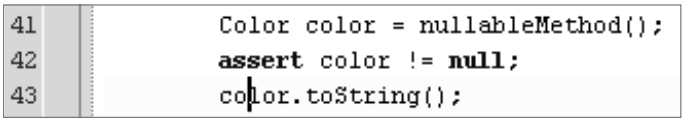
The solution is to provide the required check like below.

```
public class TestingNullable {
   @Nullable
   public Color nullableMethod(){
      //some code here

      return color;
   }

   public void usingNullableMethod(){
      // some code
      Color color = nullableMethod();

      // Introducing assurance of not-null resolves the problem
      if (color != null) {
      color.toString();
      }
   }
}
```

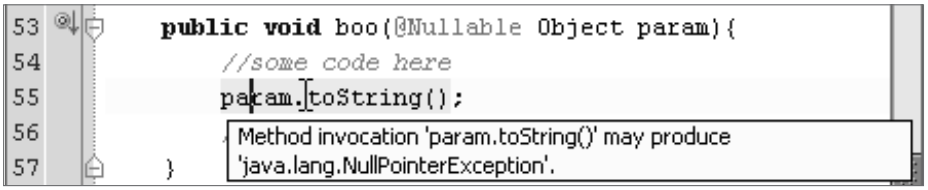
You can also use the solution suggested by the IntelliJ IDEA quick-fix.



Here is how the fixed code will look.



Similar usage is applicable for variables. For instance, if a parameter is declared with the `@Nullable` annotation, you will get the warning message if it is not appropriately checked.



And solution is similar – check for **`null`**.

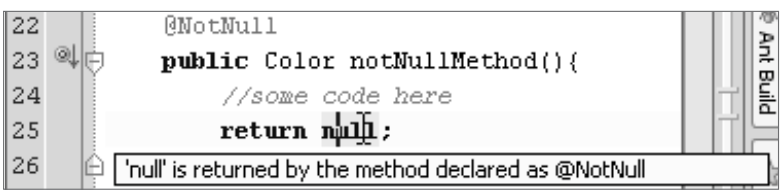
@NotNull Annotation - Examples

The `@NotNull` annotation objective is to inform you if a return of the `null` value is prohibited for the method or `null` is an illegal value for a variable.

For instance, there is the following code.

```
public class TestingNullable {
   @NotNull
   public Color notNullMethod(){
      // some code here

      // The @NotNull-annotated method returns null,
      // which is prohibited
      return null;
   }
}
```

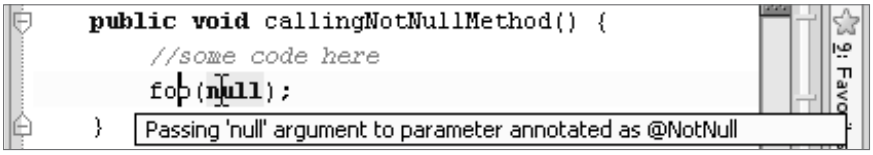
IntelliJ IDEA indicates that the method returns `null` while it is explicitly declared that it should not.



Variables are another issue. For instance, let's create a method where the parameter has the `@NotNull` annotation. Then call it using `null` as the parameter value.

```
public class TestingNullable {
   public void foo(@NotNull Object param){
      //some code here
   }

   ...

   public void callingNotNullMethod() {
      //some code here
      // the parameter value according to the explicit contract
      // cannot be null
      foo(null);
   }
}
```

IntelliJ IDEA shows the following message.



The method should be called with the correct parameter.

For this description we intentionally used rather simple and straight-forward code samples. However, IntelliJ IDEA can detect more complicated cases as well. For instance, you have a method annotated as `@Nullable`. And then use its result as a return value of the method annotated as `@NotNull`:

```
public class TestingNullable {
   @NotNull
   public Color notNullMethod(){

      // the method is highlighted - the nullableMethod()
      // method is declared as @Nullable
            return nullableMethod();
   }
}
```

Another case – calling the `foo` method with the `param` parameter which is annotated as `@NotNull`. IntelliJ IDEA informs you if in the `bar` method which calls the `foo` method, the `objParam` value is `null`.

```
public class TestingNullable {
   public void foo(@NotNull Object param){
      //some code here
   }

   public void bar(Object objParam){
```

```
        // Detecting that parameter is null - and highlighting
        // it as a contract violation
        if(objParam == null) foo(objParam);
    }
}
```

@Nullable Problems Inspection

There is a separate inspection which adds to the `@Nullable` functionality in IntelliJ IDEA. It is called **@Nullable problems**. This inspection covers several issues related to the descendants overriding/implementing annotated methods:

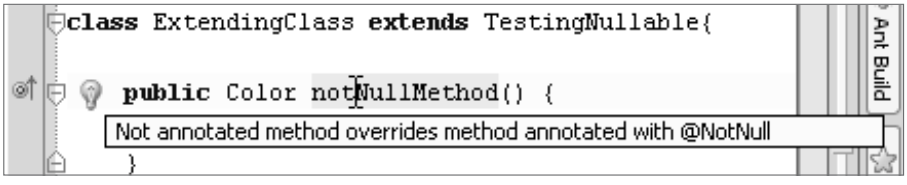
- `@NotNull parameter overrides @Nullable`

  If a parameter in the overriding method is annotated as `@NotNull` while the parent method parameter has the `@Nullable` annotation, IntelliJ IDEA warns you about that.

```
public class TestingNullable {
   public void boo(@Nullable Object param){
      //some code here
   }
}

class ExtendingClass extends TestingNullable{

   // The overriding method parameter is highlighted.
   public void boo(@NotNull Object param) {
      super.boo(param);
   }
}
```

- `Not annotated method overrides @NotNull`

  If there is a class overriding the given `@NotNull` method, IntelliJ IDEA warns you if the overriding method does not have a `@NotNull` annotation.



- `@Nullable method overrides @NotNull`

  `@Nullable`-annotated methods should not override ones annotated as `@NotNull`.