Static and dynamic typing

• Static typing
  – Compiler guarantees that some errors cannot happen
    • The set of errors depends on the language
    • Other errors are still possible!
  – Examples: C, C++, Objective C, C#, Java, Haskell, ML

• Dynamic typing
  – The run-time system keeps track of types, and throws errors
  – Examples: Lisp, Scheme, Perl, PHP, Python, Ruby, JavaScript

• No type system
  – Example: Assembly
Why we ♥ static typing

• Documentation
• Correctness/reliability
• Refactoring
• Speed
Why we ♥ dynamic typing
(= Why we ☹️ static typing)

• More concise code
  – Type inference is possible
• No false positive warnings
  Every static type system rejects some correct programs
  `@NonNull String lineSep = System.getProperty("line.separator");`
• More flexible code
  – Add fields at run time
  – Change class of an object
• Ability to run tests at any time
  – Feedback is important for quality code
  – Programmer knows whether static or dynamic feedback is best
Nullness subtyping relationship

• Which type hierarchy is best?

@NonNull Date
@Nullable Date
@?? Date

• A subtype has more values
• A subtype has more operations
• A subtype is substitutable
• A subtype preserves supertype properties
Mutability subtyping relationship

• Which type hierarchy is best?

@Immutable Date
@Mutable Date
@ReadOnly Date

@ReadOnly: no one can do mutation
@Mutable: anyone can do mutation
  - No guarantee about mutation from elsewhere
Flow sensitivity

• Which calls are legal?

```
Object name;
nname = new Object();
nname.toLowerCase();
nname = "HELLO";
nname.toLowerCase();
nname = new Object();
nname.toLowerCase();

@Nullable String name;
nname = null;
nname.toLowerCase();
nname = "HELLO";
nname.toLowerCase();
nname = null;
nname.toLowerCase();
```
Flow sensitivity: name and legality

• Control flow determines the type

```java
if (x==null) {
    ...
    // treat as nullable
} else {
    ...
    // treat as non-null
}
```

• What changes to the type are legal?

<table>
<thead>
<tr>
<th><code>String name;</code></th>
<th><code>@NonNull String name;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name = new Object();</code></td>
<td><code>name = null;</code></td>
</tr>
<tr>
<td><code>... // treat name as Object</code></td>
<td><code>... // treat name as nullable</code></td>
</tr>
</tbody>
</table>

Not these; only change to a subtype
Flow sensitivity and type inference

When must you write a type?

If the default is the *top* of the type hierarchy, you don’t need to annotate local variables

```java
@Nullable String name;
name = "hello";
...  // treat name as non-null

@Nullable String name;
name = otherNullable;
...  // treat name as nullable
```
The receiver is just another parameter

How many arguments does `Object.equals` take?

```java
class MyClass {
    @Override
    public boolean equals(Object other) { ... }
}
```

Two! Their names are `this` and `other`

Neither one is mutated by the method

```java
public boolean equals(@ReadOnly Object other) @ReadOnly { ... }
```