Type systems

CSE 331

Spring 2010

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

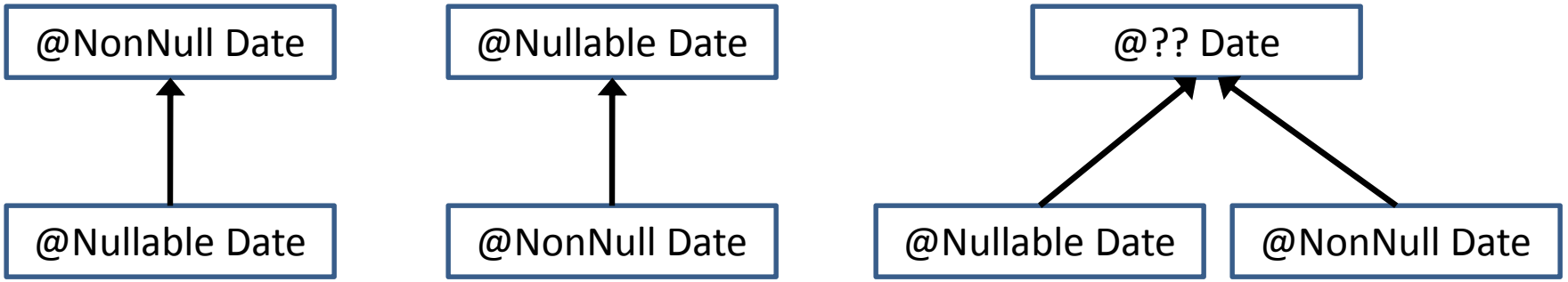
  Every static type system rejects some correct programs

```
@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

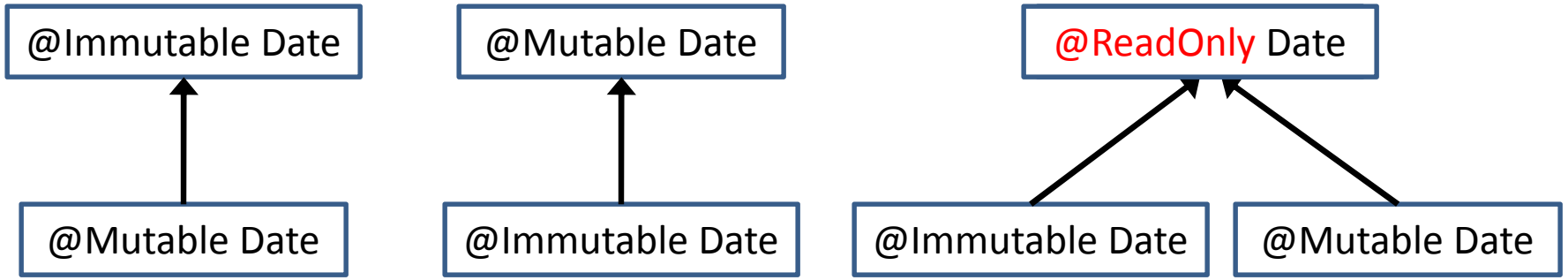
- Which type hierarchy is best?



- 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: no one can do mutation
– I can't do mutation

@ReadOnly

@Mutable: anyone can do mutation
– No guarantee about mutation from elsewhere

Flow sensitivity

- Which calls are legal?

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

```
@Nullable String name;
name = null;
name.toLowerCase();
name = "HELLO";
name.toLowerCase();
name = null;
name.toLowerCase();
```

Flow sensitivity: name and legality

- Control flow determines the type

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

- What changes to the type are legal?

```
String name;
name = new Object();
... // treat name as Object
```

```
@NonNull String name;
name = null;
... // treat name as nullable
```

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

`@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?

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

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

Neither one is mutated by the method

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

Annotation on type of `other`

Annotation on type of `this`