Java Annotations

- Attach meta-data to program constructs
  - Data about the program, not data in the program

- Formerly often specified as comments
  - Can now be checked and processed automatically
  - Avoids parsing of source code or strings
  - Annotations act as "smart comments"

- Annotations are product types ("structs") composed of constants
  - primitives (int, double, etc.)
  - enums
  - strings
  - class constants (Integer.class)
  - other annotations
  - arrays of the above

Subtyping

- Annotations in Java do not have a common supertype
  - Cannot define an annotation that can contain ANY other annotation

- Added subtyping for annotations to Java
  - Minimal changes to compiler, no changes to class file format
  - Minor changes to reflection API to support additional features

- Integrates well with existing code
  - Improves @DefaultQualifier annotation, which currently uses a string

Subtyping Examples

```
@DefaultQualifier { String value; }
@interface DefaultQualifiers { DefaultQualifier[] value; }
@interface NonNull { }
@interface Interned { }
@DefaultQualifiers({@DefaultQualifier("NonNull"),
                    @DefaultQualifier("Interned")})
class MyClass { ... }
```

```
@interface Annotation { }
@interface DefaultQualifier { Annotation[] value; }
@interface NonNull extends Annotation { }
@interface Interned extends Annotation { }
@DefaultQualifier({@NonNull, @Interned})
class MyClass { ... }
```

Invariant Specification

- Program invariants can be encoded as annotations and checked automatically
  - Similar to assert statements, but inherited into subclasses
  - Generates log file instead of terminating program
  - Simple generation of invariant index using Javadoc tool

```
interface TableModel {
    // invariant: must be called from within event thread
    @OnlyEventThread void setValueAt(...);
}
class MyTableModel implements TableModel {
    void setValueAt(...) { /* invariant automatically inherited */ }
}
```

Multi-Staging

- Multi-stage programming (MSP) is a paradigm for developing generic software without paying a runtime penalty for this generality
  - "Staging" moves computations into a code generation step before runtime
  - Generically written code (e.g. power) is optimized for special cases (e.g. square)

```
@Code double power(@Code double x, int n) {
    if (n==0) return @Code (1.0);
    else return @Code (@Escape (x) * @Escape (power(x, n-1)));
}
double square(double x) { return @Run (power(@Code (x), 2)); }
```

Additional Targets

- Annotations in Java cannot be attached to statements or expressions

- Allow additional targets for annotations
  - block statements
  - parenthetical expressions

```
@Contained { // block annotation
    // block of code that does not spawn async. tasks ("contained")
    int i = -5;
    int j = @AlwaysPositive (i*j); // paren. expression annotation
}
```