Define serialization.

Converting an object into a series of bytes. This is handy for writing to disk or transmitting over a network.

Give the annotation indicating serializability.

`@ serializable` to classes.

`@ SerialVersion UID(1234).`

This uses Serializable interface and SerualVersion UID member for JVM.

Are annotation automatic getters and setters generally needed? Why?

When are they available?

They're generally not needed because Scala blends syntax for field access and method invocation.

But some frameworks require it, so add to field:
`@scala.reflect.BeanProperty`

They are only available after compilation.

Give the annotation to indicate that a class or method should **not** be used.

`@ deprecated`

This passes on `@Deprecated` to the JVM.

What can annotation arguments be sometimes?

Arbitrary expressions, so long as they type check.

How do you add an annotation to an expression?

Place a colon after, then annotation.

```
(e:@unchecked) match {
  //non-exhaustive match
}
```

Give the annotation to indicate concurrently accessed mutable state.

What are the guarantees?

`@ volatile`

The guarantees are platform specific, but in Java the same as the volatile keyword.

Give the annotation syntax.

```
@ annot(exp1, exp2, ...) {val name = const,
..., val namen = constn}
```

The order of named values is irrelevant.

Give the annotation to indicate a non-serializable field.

What does it do?

`@ transient`

This forces non-serialization - when unserialized will get the default value for the type.

The transient keyword is used for JVM.

Where are annotations allowed?

- On any declaration or definition, including vals, vars, defs, classes, singletons, traits, type aliases.
- On expressions.
- On types.