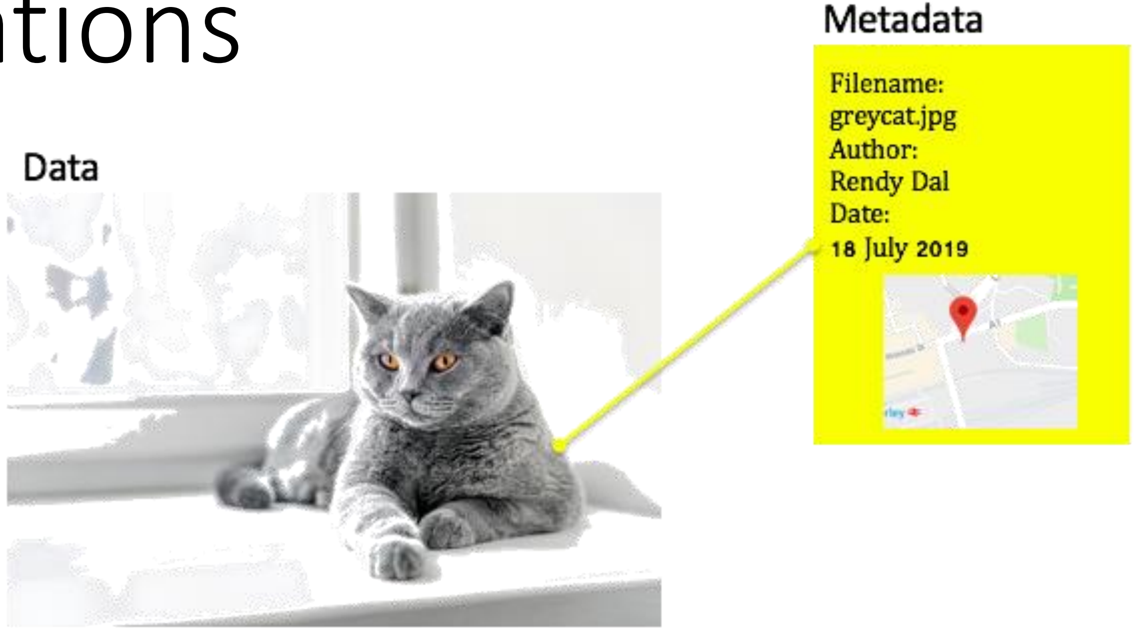
Application Design Using Java

Lecture 25

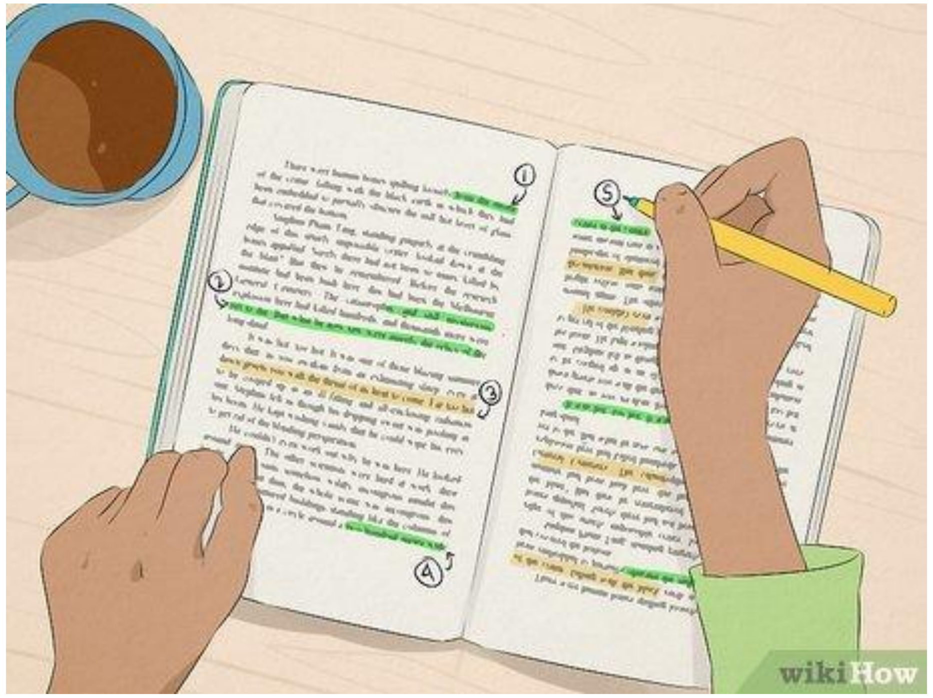
Metadata and Annotations

- Data about data
- Categories
  - Technical
  - Business
  - Process
- Annotation is a form of attaching additional metadata to code



Annotations

- Embed additional information into the source code
- Influence how programs are processed by libraries and tools
  - Compiler
  - IDE
  - Runtime tools



Annotations in Java

- Tags that you insert into your source code
- Do not change the way in which programs are compiled
- To benefit from annotations, you need a *processing tool*
- Possible use
  - Automatic generation of auxiliary files, such as deployment descriptors or bean information classes
  - Automatic generation of code for testing, logging, transaction semantics, and so on

Why Annotations?

- Declarative programming style
    - Less programming because tools can generate the required environment
    - Easier to modify
- No need to have additional files with supplementary information

Using Annotations

- Used like a *modifier* and are placed before the annotated item *without a semicolon*
- By convention, placed before all other modifiers (`public`, `static`, etc.)
- The name is preceded by an @ symbol, similar to Javadoc comments
- Element-value pairs may be provided after the name
- Annotations are part of the code unlike Javadoc comments which occur inside the /** . . . */ delimiters
- Can be applied to
  - Packages
  - Classes
  - Methods
  - Fields
  - Local variables

```
@Test(timeout="10000")
public void checkRandomInsertions()
```

Ad-hoc “Almost” Annotations

- `transient`
- Serializable interface
- @deprecated
- Javadoc tags

Defining Annotations

- `@interface` construct
- Each method defines an element of the annotation
- Methods must not have any parameters or declare exceptions they might throw
- Return types are limited
  - String
  - Class
  - Enumerations
  - Annotations
  - Arrays of any of the above types
- Methods may have default values

Types of Annotations

- Marker
  - No elements
  - Parentheses () may be omitted
- Single value
  - Element name must be `value`
  - When using, element name and = may be omitted
- Regular

```
public @interface Preliminary { }
```

```
@Preliminary public class TimeTravel { ... }
```

```
public @interface Copyright {
    String value();
}
```

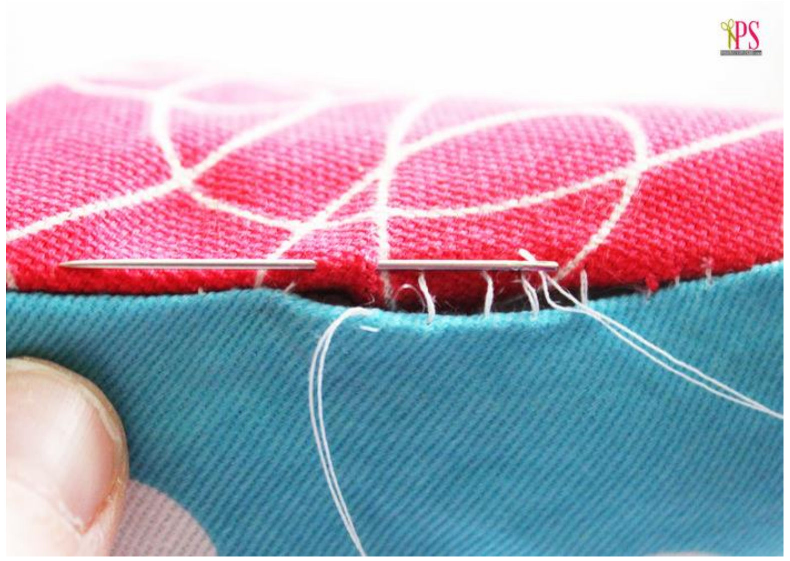
```
@Copyright("2021 Space Propulsion Systems")
public class OscillationOverthruster { ... }
```

Meta-annotations

- Used to annotate annotations
- @Retention
  - Defines for how long information from the annotation will be stored
  - Enum RetentionPolicy
    SOURCE, CLASS (Default), RUNTIME
- @Target
  - Limits the use of the annotation
  - Enum ElementType
    TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL_VARIABLE, ANNOTATION_TYPE, PACKAGE

Advantages of Annotations

- Cleaner code
- Static type checking – the compiler checks whether the annotation is applicable and how
- Attached to classes and can be processed via reflection



Processing Annotations

- Using `javap` command
- Reflection library

```
javap -c -p -v AnnotationDemo
…
SourceFile: "AnnotationDemo.java"
RuntimeVisibleAnnotations:
  0: #20(#21=I#22,#23=s#24,#25=s#26)
    RequestForEnhancement(
      id=12345
      synopsis="Enable additional functionality"
      date="4/28/2021"
    )
```

Reflection

- If a program can analyze its own capabilities it is called *reflexive*
- Reflection library is a set of tools that enables dynamic manipulation of Java code
- Potential uses
  - Analyze the capabilities of classes at runtime
  - Inspect objects at runtime
  - Implement generic array manipulation code
  - Take advantage of `Method` objects that work just like function pointers

The Class Class

- Java runtime system maintains *runtime type identification* on all objects
- Classes can now be treated as runtime objects
- `Class` is a special Java class
- `Class` is generic
- Describes the properties of a particular class
- `getClass()` method in the `Object` class returns an instance of `Class` type
- `forName()` returns an instance of `Class` type corresponding to the class with the given name

```
Class cl = Class.forName(“java.util.Random”);
```

- `T.class` represents the matching `Class` object for any type `T`

```
Class cl1 = java.util.Random.class;
Class cl2 = int.class;
Class cl3 = Double[].class;
```

Creating New Instances

- `getConstructor()` to get an object of type `Constructor`
- `newInstance()` to construct an instance

```
Class cl = Class.forName(“java.util.Random”);
Object obj =
cl.getConstructor().newInstance();
```

Reflection and Annotations

- `isAnnotationPresent()`
- `getAnnotation()`
- `value()`

```
boolean isBeta =
    MyClass.class.isAnnotationPresent(BetaVersion.class);
```

```
String copyright =
    MyClass.class.getAnnotation(Copyright.class).value();
```

```
Author author =
    MyClass.class.getAnnotation(Author.class);
String first = author.getFirst();
String last = author.getLast();
```

Application Design Using Java

- Is it the end of it?

What to Take Next I

Course	Description
CSCI-4220 Network Programming	Programming with an overview of the principles of computer networks, including an overview of the OSI reference model and various popular network protocol suites. Concentration on Unix interprocess communication (IPC), network programming using TCP and UDP, as well as client-side and mobile programming.
CSCI-496X Networking In the Linux Kernel (Communication Intensive)	Kernel organization, TCP/IP fundamentals, kernel development process, common kernel macros/data structures, kernel networking data structures, network programming basics, network utilities.
CSCI-4320 Parallel Programming and Computing	Introduction to parallel computing and programming. CUDA, MPI, MapReduce, Transactional Memory
CSCI-4380 Database Systems	Introduction to database systems, with a special emphasis on data modeling and programming.

What to Take Next II

Course	Description
CSCI-4440 Software Design and Documentation	Communicating a real-world problem and proposing a feasible solution. Transforming the real-world representation into diagrams and documentation Implementing the design and verifying through testing.
CSCI-4350 Data Science	Combines aspects of data management, library science, computer science, and physical science.
CSCI-4260 Graph Theory	Fundamental concepts of graph theory and its applications in computer, social, and natural sciences. Graphs as models; representation of graphs; trees; universal trees; distances; matchings; connectivity; flows in networks; colorings; cycles; planarity; and other computational problems and algorithms.
CSCI-496X Graph Mining	Introduction to graph processing and mining. Recommender systems, link prediction, community detection, random walks, linear algebra on graphs

What to Take Next III

Course	Description
CSCI-496X Introduction to Network Science	Interdisciplinary introduction to the emerging science of complex networks and their applications. Mathematics of networks, data analysis, network visualization, and applications to ecology, biology, sociology, technology, and other fields.
CSCI-4250 Frontiers of Network Science	Introduction to network science and review of current research in this field. Graphs and networks; random networks and various types of scale-free networks; network properties such as assortatitivity, mobility, robustness, social networks and communities; and dynamics of processes on networks.

//TODO before next lecture:

- Homework 5 due on 4/30 at 11:59 pm EDT. Must be submitted on Submitty.
- Final Team Project due on 5/3. Presentations in class starting at 12:20 pm EDT. All materials must be submitted on Submitty by 11:59 pm EDT.