Reflection

Reflection

- A program that analyzes information about classes and capabilities of those classes is *reflective*.
- Reflection allows us to inspect classes **at runtime**, without knowing the names of the classes, methods, etc. at compile time.

- Some things that were previously implemented with reflection (function pointers) can be accomplished with more user-friendly approaches in Java 8 (method references).

Using Reflection

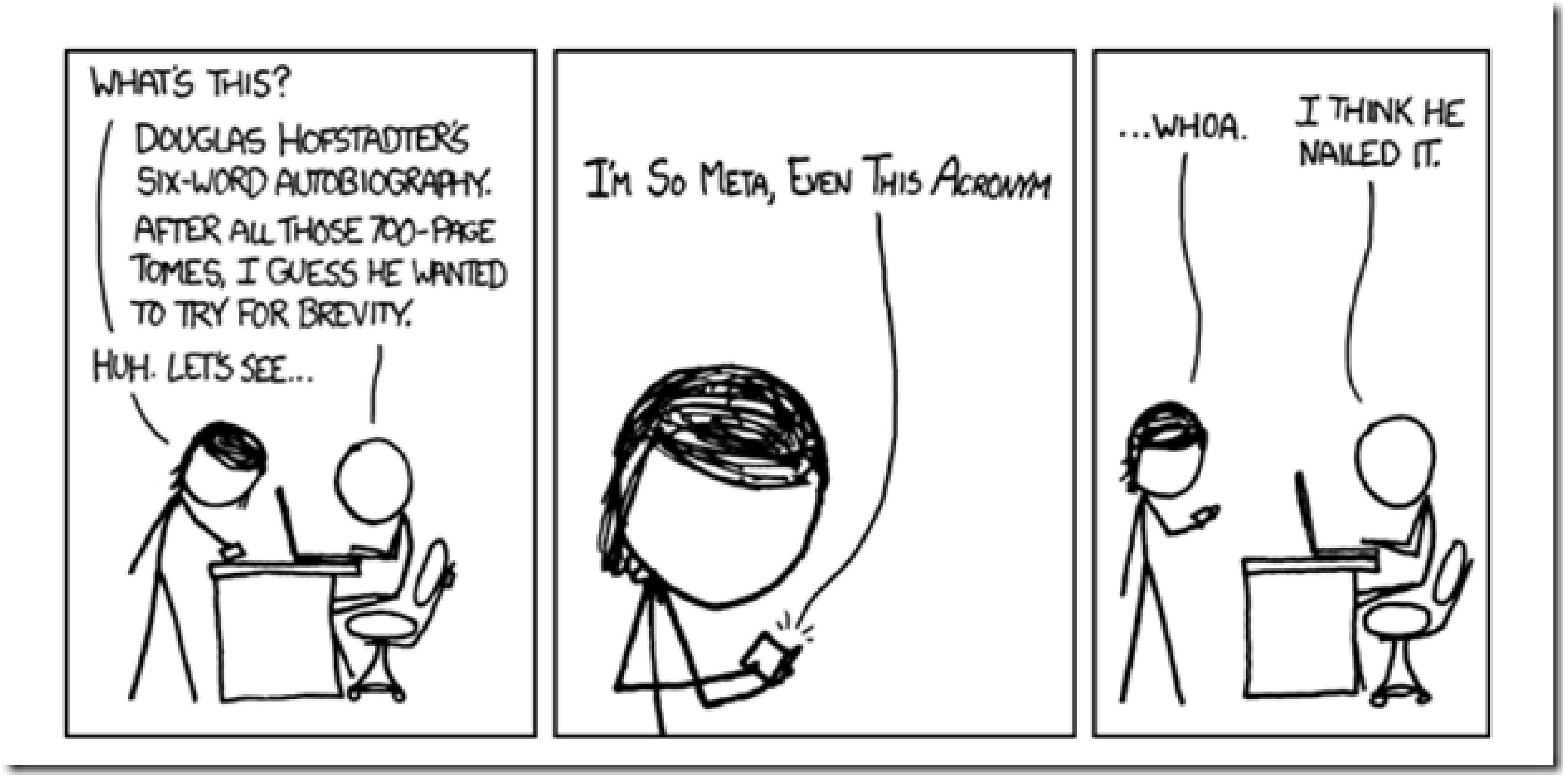
- Reflection is used in:
  - tool building (e.g., developers of IDEs)
  - unit testing
  - working with XML /JSON
  - working with a dependency injection framework
  - working with databases

- Note that reflection can also be combined with generics.

Using Reflection

- Reflection is fragile!
  - The compiler cannot help you find errors.
  - Errors occur at runtime.
- Security issues
  - Making Strings mutable?!

Think Meta!


WHAT'S THIS?
DOUGLAS HOFSTADTER'S SIX-WORD AUTOBIOGRAPHY. AFTER ALL THOSE 700-PAGE TOMES, I GUESS HE WANTED TO TRY FOR BREVITY.
HUH. LET'S SEE...
I'M SO META, EVEN THIS ACRONYM
...WHOA.
I THINK HE NAILED IT.

The `Class` Class

- The JVM maintains *runtime type identification* which is the actual type of each object.
  - This is used to figure out the correct method to call for polymorphic references.
- You can access this information at any time:

```
Class c1 = myObject.getClass();
// use this for objects

Class c2 = Employee.class;
// use this for names of classes

Class c3 = int.class;
// use this for primitive types
```

The `Class` Class

- The JVM has one unique Class object for each type, so classes can be compared with ==

```
  if(e.getClass() == Employee.class)
```

The `Class` Class

- Note: unlike instanceof, this tests whether the classes are **exactly** the same.

```
public class FullTimeEmployee extends Employee

fullTimeEmployee instanceof Employee
// true


fullTimeEmployee.getClass() == Employee.class
// false
```

`Class` Methods

- The name of the class: `getName()`
- Create a new class with static `forName` method:

```
Class cl = Class.forName(“Employee”);
```

- Create a new object with newInstance:

```
MyClass obj = theObj.getClass().newInstance();
```

  - This only works if there is a default, no-argument constructor!
- Obtain the super class: `getSuperclass()`

Other Classes in java.lang.reflect

- Method
- Constructor
- Field
  - getType
- All have
  - getName
  - getModifiers
    - Modifier class has methods isPublic, isPrivate, isFinal
    - Modifier.isStatic(field.getModifiers()) (or method.getModifiers())

Accessing Class Information

- The Class class has methods:
  - getDeclaredFields returns a Field[]
  - getDeclaredMethods returns a Methods[]
  - getDeclaredConstructors returns a Constructor[]
- Returns private, package, and protected members, but **not** members of the superclasses.

Practice

- Create some objects and print out their classes.
- Read in the type of Payer object and create a new object using newInstance.
- Use reflection to print out all of the information about the FullTimeEmployee class.
  - Fields, methods, constructors
  - Same information about its parent class(es)
- Use reflection to invoke a method on the Payer object.