REFLECTIONS: PRINCIPLES & CAVEATS

REFLECTION

CREATING AN OBJECT OF A CLASS IS CALLED **INSTANTIATION**

THE USUAL WAY TO INSTANTIATE A CLASS
WOULD BE WITH A LINE OF CODE LIKE THIS

```
ArrayList someList = new ArrayList();
```

BUT IT IS ALSO POSSIBLE TO INSTANTIATE
AN OBJECT FROM THE NAME OF THE CLASS,
USING CODE LIKE THIS

THE TECHNIQUES USED TO CREATE
AND DO STUFF WITH CLASSES AND OBJECTS
AT RUN-TIME, ON-THE-FLY, IS CALLED
REFLECTION

```
ArrayList onTheFlyList = (ArrayList)
        (Class.forName("java.util.ArrayList").newInstance());
```

THIS LINE IS A WAY TO DECIDE, ON THE FLY,
WHAT CLASS OF OBJECT YOU ARE SEEKING
TO CREATE

REFLECTION AND TYPE INTROSPECTION

THESE TWO TERMS ARE USED PRETTY MUCH INTERCHANGEABLY, BUT THEY ACTUALLY REFER TO SLIGHTLY DIFFERENT CONCEPTS

REFLECTION IS THE ABILITY, AT RUNTIME, TO ACTUALLY CREATE OBJECTS OF CLASSES, INVOKE METHODS, MANIPULATE METADA

"CREATE AN OBJECT OF CLASS FOO"

"INVOKE METHODS ON IT"

"ACCESS PRIVATE MEMBERS, EVEN FROM A THIRD PARTY JAR"

TYPE INTROSPECTION IS THE ABILTY, AT RUNTIME, TO EXPLORE THE TYPE OF AN OBJECT

"WHAT CLASS IS THIS OBJECT?"

"DOES IT SATISFY A CERTAIN INTERFACE?"

"WHAT ARE ITS MEMBER FUNCTIONS?"

REFLECTION IS VERY POWERFUL,
BUT HAS SIGNIFICANT ISSUES
THAT YOU SHOULD BE AWARE OF

COMPLEXITY

PERFORMANCE OVERHEAD

SECURITY CONSIDERATIONS

VIOLATION OF ABSTRACTIONS

COMPLEXITY

EVEN SIMPLE OPERATIONS CAN BE UNEXPECTEDLY COMPLEX USING REFLECTION

```
ArrayList onTheFlyList = (ArrayList)
    (Class.forName("java.util.ArrayList").newInstance());
```

NOTICE ALSO HOW WE HAD TO USE THE FULLY QUALIFIED CLASSNAME, ELSE A CLASSNOTFOUND EXCEPTION WOULD HAVE RESULTED

GOING BACK TO OUR EXAMPLE..

NOTICE HOW WE INSTANTIATED AN ARRAYLIST, AND NOT AN ARRAYLIST<STRING>?

THAT'S BECAUSE INSTANTIATING GENERICS USING REFLECTION IS SURPRISINGLY CONVOLUTED

(AND WOULD HAVE OBSCURED THE POINT OF THE EXAMPLE!)

PERFORMANCE OVERHEAD

(REFLECTION SHOULD NOT BE USED FOR PERFORMANCE-CRITICAL CODE)

BY DEFINITION, REFLECTION OPERATES PURELY AT

RUNTIME

NOW THE JAVA VIRTUAL MACHINE, LIKE ALL COMPILERS, DOES A BUNCH OF OPTIMIZATIONS TO IMPROVE CODE PERFORMANCE - THESE ALL HAPPEN AT

COMPILE-TIME

SO - ANYTHING DONE WITH REFLECTION IS A LOT SLOWER THAN IF DONE THE USUAL WAY

SECURITY CONSIDERATIONS

THE RUNTIME, ON-THE-FLY NATURE OF REFLECTION-BASED CODE IS WHAT MAKES IT FLEXIBLE..

..BUT THIS ALSO MEANS THAT ITS IMPOSSIBLE TO KNOW IN ADVANCE WHAT THE CODE IS ATTEMPTING TO DO

(FOR INSTANCE APPLETS, JAVA CODE RUNNING THE BROWSER, HAVE TIGHT RUNTIME SECURITY RESTRICTIONS)

FOR THIS REASON, REFLECTION-BASED CODE NEEDS ELEVATED RUNTIME PERMISSIONS, AND NOT ALL RUNTIME ENVIRONMENTS MIGHT ALLOW THIS

VIOLATION OF ABSTRACTIONS

ABSTRACTION IS A KEY UNDERLYING PRINCIPLE OF OBJECT-ORIENTED SOFTWARE DESIGN

YOU DON'T KNOW, OR NEED TO KNOW, HOW AN INTERFACE IS IMPLEMENTED, YOU JUST KNOW (AND NEED TO KNOW) WHAT METHODS THE INTERFACE EXPOSES

INTERFACE-DRIVEN PROGRAMMING, FOR INSTANCE, IS ALL ABOUT ABSTRACTION -

REFLECTION, BY GIVING UNFETTERED ACCESS TO IMPLEMENTATION DETAILS (E.G. PRIVATE MEMBER VARIABLES) CAN KILL ABSTRACTIONS AND LEAD TO POOR CODE DESIGN