# REFLECTION AND THE FACTORY PATTERN

HAVE COME TO BE QUITE CLOSELY
ASSOCIATED - SO ITS WORTH
UNDERSTANDING REFLECTION

# 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

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

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

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