reflection in Java

main features...

- inspect classes & methods at run-time
- dynamically create new objects
- dynamically invoke methods

so what…

- can get at classes from their names
- can get at methods from their names

does this replace method-encapsulating patterns…

- generally no, but in some situations…
- …it can do

creating an object #1 (zero arg constructor)

- get class from a class name

```
Class.forName( className )
```

- make an instance from a class name (constructor with no args)

```
instance = Class.forName( className ).newInstance();
```

creating an object #2 (constructor with String arg)

- make an instance from a class name (constructor with String arg)

```
Class cl = Class.forName(className);

Constructor cons = cl.getConstructor( String.class );;

instance = cons.newInstance( argAsString );
```

more generally…

```
Class…
Constructor<T> getConstructor(Class<?>... parameterTypes)
Constructor<?>[] getConstructors()

Constructor…
T newInstance(Object... initargs)
```

calling a method #1 (no args)

1. get the class

```
// getting the class from the class name
Class cl = Class.forName(className);

// getting the class from the instance
Class cl = instance.getClass();
```

2. get the method (no args) & invoke it

```
Method m = cl.getDeclaredMethod( methodName );

result = m.invoke( classInstance );
```

continued…

calling a method #2 (using args)

1. get the class – same as before

2. get the method (1 string arg) & invoke it

```
Method m = cl.getDeclaredMethod(methodName,String.class);

result = m.invoke( classInstance, argValue );
```

more generally…

```
Class…
Method getDeclaredMethod(String name,
                         Class<?>... parameterTypes)

Method[]   getDeclaredMethods()

Method…
Object  invoke(Object obj, Object... args)
```