Reflection in Java

CMSC132 - Fall 2020
Object-Oriented Programming II

Built using as reference:
1. Manuel Oriol's (ETH) slides on “Reflection in Java”
2. Kim Mens' (UCL) slides on “Basics of Reflections in Java”
3. upen.rockin slides on “Reflection in Java”
A Few Concepts

Reflection

The ability of a program to inspect and change itself during runtime.
A Few Concepts

Reflection
The ability of a program to inspect and change itself during runtime.

Reification
Making the language concepts accessible to the program, to be manipulated as ordinary data.
A Few Concepts

**Reflection**
The ability of a program to inspect and change itself during runtime.

**Introspection**
Self-examination - only looking at the reified entities.

**Reification**
Making the language concepts accessible to the program, to be manipulated as ordinary data.
A Few Concepts

Reflection
The ability of a program to inspect and change itself during runtime.

Introspection
Self-examination - only looking at the reified entities.

Reification
Making the language concepts accessible to the program, to be manipulated as ordinary data.

Intercession
Using introspection to intervene in the program execution, by manipulating the reified entities.
Reflection

*It is like...*
Reflection
*It is like...*

“Consciousness”
for Java classes
How?

```java
package blackjack;
import java.util.*;
public class Blackjack implements BlackJackEngine {
    private ArrayList<Card> gameDeck, dealerHand, playerHand;
    private int gameStatus;
    private int betAmount;
    private int account;
    private static final int INITIAL_ACCOUNT_VALUE = 200;
    private static final int INITIAL_BET_AMOUNT = 5;
    private Random randomGenerator;
    private int numberOfDecks;

    /**
     * Constructor you must provide. Initializes the player's account
     * to 200.00 and the initial bet to 5. Feel free to initialize any other
     * fields. Keep in mind that the constructor does not define the
     * deck(s) of cards.
     * @param randomGenerator
     * @param numberOfDecks
     */
    public Blackjack(Random randomGenerator, int numberOfDecks) {
        this.randomGenerator = randomGenerator;
        this.numberOfDecks = numberOfDecks;
        betAmount = INITIAL_BET_AMOUNT;
        account = INITIAL_ACCOUNT_VALUE;
    }

    public int getNumberOfDecks() {
        return numberOfDecks;
    }

    public void createAndShuffleGameDeck() {
        gameDeck = createAndShuffleGameDeckInternal(numberOfDecks);
    }

    public Card[] getGameDeck() {
        Card[] result = new Card[gameDeck.size()];
        gameDeck.toArray(result);
        return result;
    }
```
public class Blackjack implements BlackjackEngine {
    private String[] gameDeck, dealerHand, playerHand;
    private int gameStatus;
    private int betAmount;
    private int account;
    private static final int INITIAL_ACCOUNT_VALUE = 200;
    private static final int INITIAL_BET_AMOUNT = 5;
    private Random randomGenerator;
    private int numberOfDecks;

    /**
     * Constructor you must provide. Initializes the player's account
     * to 200.00 and the initial bet to 5. Feel free to initialize any other
     * fields. Keep in mind that the constructor does not define the
     * deck(s) of cards.
     * @param randomGenerator
     * @param numberOfDecks
     */
    public Blackjack(Random randomGenerator, int numberOfDecks) {
        this.randomGenerator = randomGenerator;
        this.numberOfDecks = numberOfDecks;
        betAmount = INITIAL_BET_AMOUNT;
        account = INITIAL_ACCOUNT_VALUE;
    }

    public int getNumberOfDecks() {
        return numberOfDecks;
    }

    public void createAndShuffleGameDeck() {
        gameDeck = createAndShuffleGameDeckInternal(numberOfDecks);
    }

    public Card[] getGameDeck() {
        Card[] result = new Card[gameDeck.size()];
        gameDeck.toArray(result);
        return result;
    }
}
How?

```java
package blackjack;
import java.util.*;

public class Blackjack implements BlackjackEngine {
    private ArrayList<Card> gameDeck, dealerHand, playerHand;
    private int gameStatus;
    private int betAmount;
    private int account;
    private static final int INITIAL_ACCOUNT_VALUE = 200;
    private static final int INITIAL_BET_AMOUNT = 5;
    private Random randomGenerator;
    private int numberOfDecks;

    /**
     * Constructor you must provide. Initializes the player's account
     * to 200.00 and the initial bet to 5. Feel free to initialize any other
     * fields. Keep in mind that the constructor does not define the
     * deck(s) of cards.
     * @param randomGenerator
     * @param numberOfDecks
     */
    public Blackjack(Random randomGenerator, int numberOfDecks) {
        this.randomGenerator = randomGenerator;
        this.numberOfDecks = numberOfDecks;
        betAmount = INITIAL_BET_AMOUNT;
        account = INITIAL_ACCOUNT_VALUE;
    }

    public int getNumberOfDecks() {
        return numberOfDecks;
    }

    public void createAndShuffleGameDeck() {
        gameDeck = createAndShuffleGameDeckInternal(numberOfDecks);
    }

    public Card[] getGameDeck() {
        Card[] result = new Card[gameDeck.size()];
        gameDeck.toArray(result);
        return result;
    }
}
```

In the package java.lang.reflect
How?

```java
package blackjack;
import java.util.*;

public class Blackjack implements BlackjackEngine {
    private ArrayList<Card> gameDeck, dealerHand, playerHand;
    private int gameStatus;
    private int betAmount;
    private int account;
    private static final int INITIAL_ACCOUNT_VALUE = 200;
    private static final int INITIAL_BET_AMOUNT = 5;
    private Random randomGenerator;
    private int numberOfDecks;

    /**
     * Constructor you must provide. Initializes the player's account
     * to 200.00 and the initial bet to 5. Feel free to initialize any other
     * fields. Keep in mind that the constructor does not define the
     * deck(s) of cards.
     * @param randomGenerator
     * @param numberOfDecks
     */
    public Blackjack(Random randomGenerator, int numberOfDecks) {
        this.randomGenerator = randomGenerator;
        this.numberOfDecks = numberOfDecks;
        betAmount = INITIAL_BET_AMOUNT;
        account = INITIAL_ACCOUNT_VALUE;
    }

    public int getNumberOfDecks() {
        return numberOfDecks;
    }

    public void createAndShuffleGameDeck() {
        gameDeck = createAndShuffleGameDeckInternal(numberOfDecks);
    }

    public Card[] getGameDeck() {
        Card[] result = new Card[gameDeck.size()];
        gameDeck.toArray(result);
        return result;
    }
}
```

In the package java.lang.reflect
How?

```java
package blackjack;
import java.util.*;

public class Blackjack implements BlackjackEngine {
    private ArrayList<Card> gameDeck, dealerHand, playerHand;
    private int gameStatus;
    private int betAmount;
    private int account;
    private static final int INITIAL_ACCOUNT_VALUE = 200;
    private static final int INITIAL_BET_AMOUNT = 5;
    private Random randomGenerator;
    private int numberOfDecks;

    /**
     * Constructor you must provide. Initializes the player's account
     * to 200.00 and the initial bet to 5. Feel free to initialize any other
     * fields. Keep in mind that the constructor does not define the
     * deck(s) of cards.
     * @param randomGenerator
     * @param numberOfDecks
     */
    public Blackjack(Random randomGenerator, int numberOfDecks) {
        this.randomGenerator = randomGenerator;
        this.numberOfDecks = numberOfDecks;
        betAmount = INITIAL_BET_AMOUNT;
        account = INITIAL_ACCOUNT_VALUE;
    }

    public int getNumberOfDecks() {
        return numberOfDecks;
    }

    public void createAndShuffleGameDeck() {
        gameDeck = createAndShuffleGameDeckInternal(numberOfDecks);
    }

    public Card[] getGameDeck() {
        Card[] result = new Card[gameDeck.size()];
        gameDeck.toArray(result);
        return result;
    }
}
```

In the package `java.lang.reflect`
How?

In the package `java.lang.reflect`
Basically...
Basicallly...

Classes to describe
all other classes
Every class is associated with a Class object...

And every object belongs to a class.
- Ah?
- Let's just use an example...
class Class

In java.lang.Class

Every class is associated with a Class object...

And every object belongs to a class.
- Ah?
- Let's just use an example...

```java
Class c = "foo".getClass();
System.out.print(c.getName());

// This prints "String"
```
class Class

Every class is associated with a Class object...

And every object belongs to a class.
- Ah?
- Let's just use an example...

```java
Class c = "foo".getClass();
System.out.print(c.getName());

// This prints "String"
```

Objects of class String
class **Class**

*In* java.lang.Class

Every class is associated with a Class object...

And every object belongs to a class.
- Ah?
- Let's just use an example...

```java
Class c = "foo".getClass();
System.out.print(c.getName());

// This prints "String"
```

Objects of class String

Objects of class Class

"Integer"

"foo"

"String"

"Hello world"

Integer

Shape

String

Double

Scanner
**class** Class

_In java.lang.Class_

Every class is associated with a Class object...

And every object belongs to a class.
- Ah?
- Let's just use an example...

```java
Class c = "foo".getClass();
System.out.print(c.getName());

// This prints "String"
```

Objects of class String

Objects of class Class
**class Class**  
*In java.lang.Class*

Every class is associated with a Class object...

And every object belongs to a class.
- Ah?
- Let's just use an example...

```java
Class c = "foo".getClass();
System.out.println(c.getName());

// This prints "String"
```

![Diagram](image-url)
Every class is associated with a Class object...

And every object belongs to a class.
- Ah?
- Let's just use an example...

```java
Class c = "foo".getClass();
System.out.println(c.getName());
// This prints "String"
```

Objects of class String

"Hello world" → getName() → "String"

Objects of class Class

8 10 -3
57 1

Objects of class Integer

Integer
Scanner
Shape
String
Double
**class** Class

*In java.lang.Class*

Every class is associated with a Class object...

And every object belongs to a class.
- Ah?
- Let's just use an example...

```java
Class c = "foo".getClass();
System.out.print(c.getName());

// This prints "String"
```

Objects of class String

Objects of class Class

```
Integer
"foo"
"String"
"Hello world"
```

```
Integer
Shape
Scanner
String
Double
```

```
8
10
-3
57
1
```

```
Objects of class Integer
```

```
Objects of class Class
```

```
GetName()
```

```
GetClass()
```
### Class

In `java.lang.Class`

Every class is associated with a Class object...

And every object belongs to a class.

- Ah?
- Let's just use an example...

```java
Class c = "foo".getClass();
System.out.println(c.getName());

// This prints "String"
```

Objects of class `Class`

Objects of class `String`

Objects of class `Integer`
# Class `Class<T>`

In java.lang.Class

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>static <code>Class&lt;T&gt;</code></td>
<td><code>forName(String name)</code></td>
</tr>
<tr>
<td><code>String</code></td>
<td><code>getName()</code></td>
</tr>
<tr>
<td><code>T</code></td>
<td><code>cast(Object obj)</code></td>
</tr>
<tr>
<td><code>boolean</code></td>
<td><code>isInstance(Object obj)</code></td>
</tr>
<tr>
<td><code>Class&lt;?&gt;[]</code></td>
<td><code>getClasses()</code></td>
</tr>
<tr>
<td><code>Field</code></td>
<td><code>getField(String name)</code></td>
</tr>
<tr>
<td><code>Constructor&lt;T&gt;</code></td>
<td><code>getConstructor(...)</code></td>
</tr>
<tr>
<td><code>Method</code></td>
<td><code>getMethod(String name, ...)</code></td>
</tr>
<tr>
<td><code>Field[]</code></td>
<td><code>getFields()</code></td>
</tr>
<tr>
<td><code>Constructor&lt;?&gt;[]</code></td>
<td><code>getConstructors()</code></td>
</tr>
<tr>
<td><code>Method[]</code></td>
<td><code>getMethods()</code></td>
</tr>
</tbody>
</table>
**class Field**

*In the java.lang.reflect package*

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>getName()</td>
</tr>
<tr>
<td>Class</td>
<td>getType()</td>
</tr>
<tr>
<td>Object</td>
<td>get(Object obj)</td>
</tr>
<tr>
<td>int / double / ...</td>
<td>getInt / getDouble / ...</td>
</tr>
<tr>
<td>void</td>
<td>set(Object obj, Object value)</td>
</tr>
<tr>
<td>void</td>
<td>setInt / setDouble / ...</td>
</tr>
</tbody>
</table>
class Field
In the java.lang.reflect package

Circle c = new Circle();

...  

c.radius = 1.23;

// This is equivalent to doing the following:

(c.getClass()).getField("radius").setDouble(c,1.23);
**class Field**

*In the java.lang.reflect package*

```java
Circle c = new Circle();
...

c.radius = 1.23;

// This is equivalent to doing the following:
(c.getClass()).getField("radius").setDouble(c,1.23);
```

Note that the `Class` object associated with `Circle` isn’t related to object `c`. We only used `c` to get the `Class` object, hence, we need to pass `c` as parameter in `setDouble`. 
**class Field**

*In the java.lang.reflect package*

```java
Circle c = new Circle();

...

c.radius = 1.23;

// This is equivalent to doing the following:

(c.getClass()).getField("radius").setDouble(c, 1.23);
```

Note that the `Class` object associated with `Circle` isn't related to object `c`. We only used `c` to get the `Class` object, hence, we need to pass `c` as parameter in `setDouble`. 
class Field

In the java.lang.reflect package

Circle c = new Circle();

...

c.radius = 1.23;

// This is equivalent to doing the following:

(c.getClass()).getField("radius").setDouble(c, 1.23);

Note that the Class object associated with Circle isn't related to object c. We only used c to get the Class object, hence, we need to pass c as parameter in setDouble.
**class** Constructor

*In the java.lang.reflect package*

<table>
<thead>
<tr>
<th>String</th>
<th>getName()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>getDeclaringClass()</td>
</tr>
<tr>
<td>int</td>
<td>getParameterCount()</td>
</tr>
<tr>
<td>Class&lt;?&gt;[]</td>
<td>getParameterTypes()</td>
</tr>
<tr>
<td>T</td>
<td>newInstance( ... )</td>
</tr>
</tbody>
</table>
class Constructor
In the java.lang.reflect package

Circle c = new Circle(1.23);

// This is equivalent to doing the following:

Circle c = (Circle.class).getConstructor(double.class).newInstance(1.23);
class Constructor

In the java.lang.reflect package

Circle c = new Circle(1.23);

// This is equivalent to doing the following:

Circle c = (Circle.class).getConstructor(double.class).newInstance(1.23);

This isn’t that interesting...
class Constructor

In the java.lang.reflect package

Object c = new Circle(1.23);

... // Suppose that Circle c is declared in some external Library.
   // You have no access to this class, or don’t even know where
   // it’s defined. You can still use it...
   ...

Object o = new Circle(2.01749);

// The top example isn’t possible without explicitly knowing
// about the Circle class. Yet, with reflection you can still
// do it without knowing this...

Object o = (c.class).getConstructor(double.class).newInstance(2.01749);
### class Method

*In the java.lang.reflect package*

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>getName()</td>
</tr>
<tr>
<td>Class</td>
<td>getDeclaringClass()</td>
</tr>
<tr>
<td>int</td>
<td>getParameterCount()</td>
</tr>
<tr>
<td>Class&lt;??&gt;[]</td>
<td>getParameterTypes()</td>
</tr>
<tr>
<td>Class&lt;??&gt;</td>
<td>getReturnType()</td>
</tr>
<tr>
<td>Object</td>
<td>invoke(Object obj, ...)</td>
</tr>
</tbody>
</table>


class Method

In the java.lang.reflect package

t.test()

// This is equivalent to doing the following:

t.getClass().getMethod("test").invoke(t);
But... Why??

*It solves problems within OOP*

1. Flexibility

2. Extensibility

3. Pluggability
But... Why??

*It solves problems within OOP*

1. Flexibility
2. Extensibility
3. Pluggability
4. Debugging
A Simple Example

To answer the “But... Why??”

```java
public static Shape getFactoryShape(String s) {
    Shape temp = null;
    if (s.equals("Circle"))
        temp = new Circle();
    else if (s.equals("Triangle"))
        temp = new Triangle();
    else if (s.equals("Square"))
        temp = new Square();
    ...
    // Many types of shapes
    ...
    return temp;
}
```
A Simple Example

To answer the “But... Why??”

```java
public static Shape getFactoryShape(String s) {
    Shape temp = null;
    if (s.equals("Circle"))
        temp = new Circle();
    else if (s.equals("Triangle"))
        temp = new Triangle();
    else if (s.equals("Square"))
        temp = new Square();
    ...
    // Many types of shapes
    ...
    return temp;
}
```
public static Shape getFactoryShape(String s) {
    Shape temp = null;
    try {
        temp = (Shape) Class.forName(s) .getDeclaredConstructor().newInstance();
    } catch (Exception e) {} 
    return temp;
}
Now let’s consider a more ...

“Complex” Example

Open Eclipse!
Files RentCar.java & RentCarReflection.java
Performance

The elephant in the room...

t.test()
?

\[ \text{t.getClass().getMethod("test").invoke(t);} \]
Performance

The elephant in the room...

```java
p.test()
```

```java
p.getClass().getMethod("test").invoke(p);
```

Go test it in Eclipse!
Thanks!
Any Questions?