DEPARTMENT OF
COMPUTER SCIENCE

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;
    }
```

How?

class **Class**

```
1  package blackjack;
2  import java.util.*;
3
4  public class Blackjack implements BlackjackEngine {
5      private ArrayList<Card> gameDeck, dealerHand, playerHand;
6      private int gameStatus;
7      private int betAmount;
8      private int account;
9      private static final int INITIAL_ACCOUNT_VALUE = 200;
10     private static final int INITIAL_BET_AMOUNT = 5;
11     private Random randomGenerator;
12     private int numberOfDecks;
13
14     /**
15      * Constructor you must provide.  Initializes the player's account
16      * to 200.00 and the initial bet to 5.  Feel free to initialize any other
17      * fields. Keep in mind that the constructor does not define the
18      * deck(s) of cards.
19      * @param randomGenerator
20      * @param numberOfDecks
21      */
22     public Blackjack(Random randomGenerator, int numberOfDecks) {
23         this.randomGenerator = randomGenerator;
24         this.numberOfDecks = numberOfDecks;
25         betAmount = INITIAL_BET_AMOUNT;
26         account = INITIAL_ACCOUNT_VALUE;
27     }
28
29     public int getNumberOfDecks() {
30         return numberOfDecks;
31     }
32
33     public void createAndShuffleGameDeck() {
34         gameDeck = createAndShuffleGameDeckInternal(numberOfDecks);
35     }
36
37     public Card[] getGameDeck() {
38         Card[] result = new Card[gameDeck.size()];
39         gameDeck.toArray(result);
40
41         return result;
42     }
```

How?

class **Class**

```
1  package blackjack;
2  import java.util.*;
3
4  public class Blackjack implements BlackjackEngine {
5      private ArrayList<Card> gameDeck, dealerHand, playerHand;
6      private int gameStatus;
7      private int betAmount;
8      private int account;
9      private static final int INITIAL_ACCOUNT_VALUE = 200;
10     private static final int INITIAL_BET_AMOUNT = 5;
11     private Random randomGenerator;
12     private int numberOfDecks;
13
14     /**
15      * Constructor you must provide.  Initializes the player's account
16      * to 200.00 and the initial bet to 5.  Feel free to initialize any other
17      * fields. Keep in mind that the constructor does not define the
18      * deck(s) of cards.
19      * @param randomGenerator
20      * @param numberOfDecks
21      */
22     public Blackjack(Random randomGenerator, int numberOfDecks) {
23         this.randomGenerator = randomGenerator;
24         this.numberOfDecks = numberOfDecks;
25         betAmount = INITIAL_BET_AMOUNT;
26         account = INITIAL_ACCOUNT_VALUE;
27     }
28
29     public int getNumberOfDecks() {
30         return numberOfDecks;
31     }
32
33     public void createAndShuffleGameDeck() {
34         gameDeck = createAndShuffleGameDeckInternal(numberOfDecks);
35     }
36
37     public Card[] getGameDeck() {
38         Card[] result = new Card[gameDeck.size()];
39         gameDeck.toArray(result);
40
41         return result;
42     }
```

In the package java.lang.reflect

How?

class **Class**

class **Field**

```
1  package blackjack;
2  import java.util.*;
3
4  public class Blackjack implements BlackjackEngine {
5      private ArrayList<Card> gameDeck, dealerHand, playerHand;
6      private int gameStatus;
7      private int betAmount;
8      private int account;
9      private static final int INITIAL_ACCOUNT_VALUE = 200;
10     private static final int INITIAL_BET_AMOUNT = 5;
11     private Random randomGenerator;
12     private int numberOfDecks;
13
14     /**
15      * Constructor you must provide.  Initializes the player's account
16      * to 200.00 and the initial bet to 5.  Feel free to initialize any other
17      * fields. Keep in mind that the constructor does not define the
18      * deck(s) of cards.
19      * @param randomGenerator
20      * @param numberOfDecks
21      */
22     public Blackjack(Random randomGenerator, int numberOfDecks) {
23         this.randomGenerator = randomGenerator;
24         this.numberOfDecks = numberOfDecks;
25         betAmount = INITIAL_BET_AMOUNT;
26         account = INITIAL_ACCOUNT_VALUE;
27     }
28
29     public int getNumberOfDecks() {
30         return numberOfDecks;
31     }
32
33     public void createAndShuffleGameDeck() {
34         gameDeck = createAndShuffleGameDeckInternal(numberOfDecks);
35     }
36
37     public Card[] getGameDeck() {
38         Card[] result = new Card[gameDeck.size()];
39         gameDeck.toArray(result);
40
41         return result;
42     }
```

In the package java.lang.reflect

How?

class **Class**

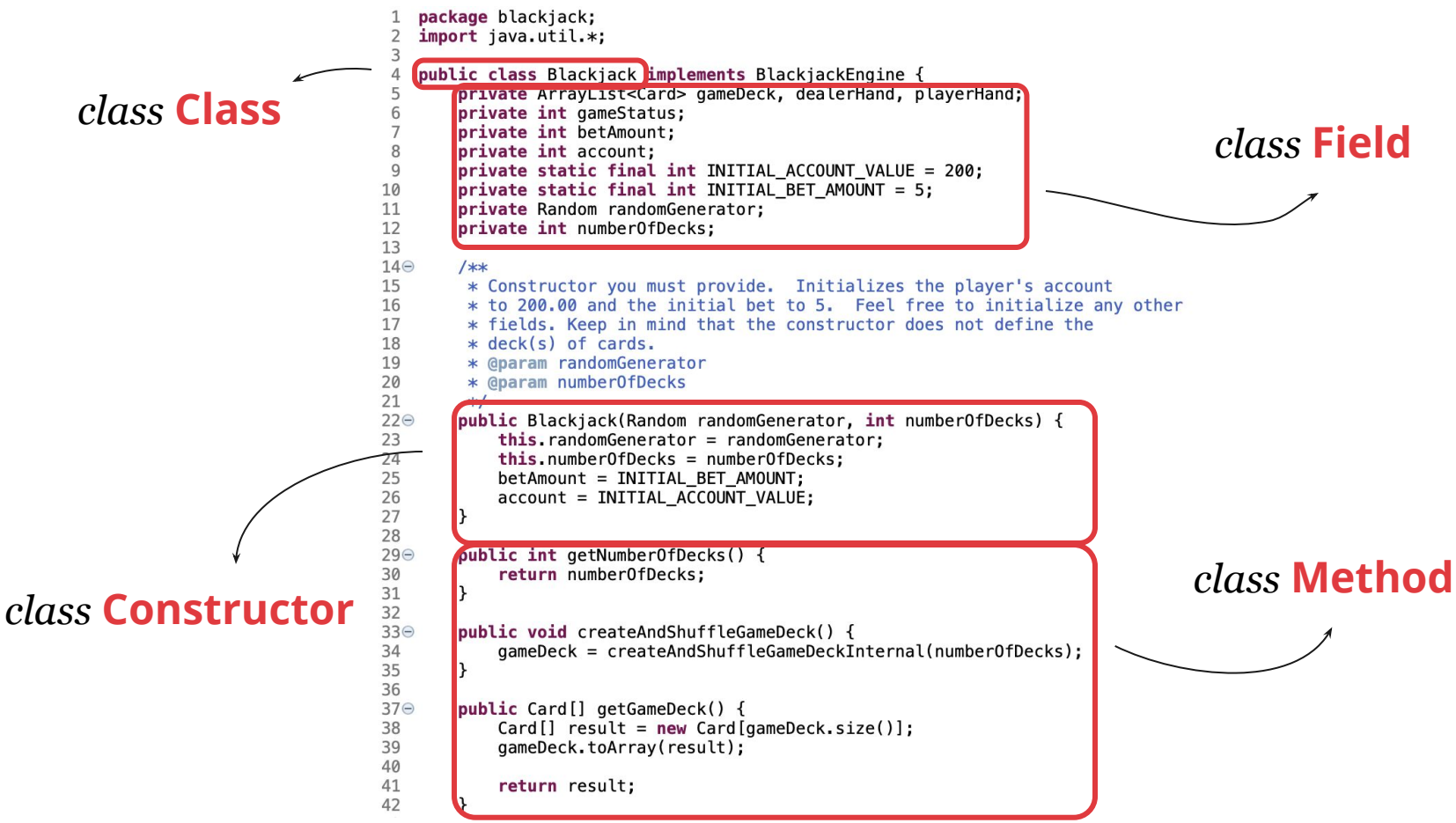
class **Field**

class **Constructor**

```
1  package blackjack;
2  import java.util.*;
3
4  public class Blackjack implements BlackjackEngine {
5      private ArrayList<Card> gameDeck, dealerHand, playerHand;
6      private int gameStatus;
7      private int betAmount;
8      private int account;
9      private static final int INITIAL_ACCOUNT_VALUE = 200;
10     private static final int INITIAL_BET_AMOUNT = 5;
11     private Random randomGenerator;
12     private int numberOfDecks;
13
14     /**
15      * Constructor you must provide.  Initializes the player's account
16      * to 200.00 and the initial bet to 5.  Feel free to initialize any other
17      * fields. Keep in mind that the constructor does not define the
18      * deck(s) of cards.
19      * @param randomGenerator
20      * @param numberOfDecks
21      */
22     public Blackjack(Random randomGenerator, int numberOfDecks) {
23         this.randomGenerator = randomGenerator;
24         this.numberOfDecks = numberOfDecks;
25         betAmount = INITIAL_BET_AMOUNT;
26         account = INITIAL_ACCOUNT_VALUE;
27     }
28
29     public int getNumberOfDecks() {
30         return numberOfDecks;
31     }
32
33     public void createAndShuffleGameDeck() {
34         gameDeck = createAndShuffleGameDeckInternal(numberOfDecks);
35     }
36
37     public Card[] getGameDeck() {
38         Card[] result = new Card[gameDeck.size()];
39         gameDeck.toArray(result);
40
41         return result;
42     }
```

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

Basically...

Basically...

Classes to describe

all other classes

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...

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...

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

// This prints "String"
```

class Class

In java.lang.Class

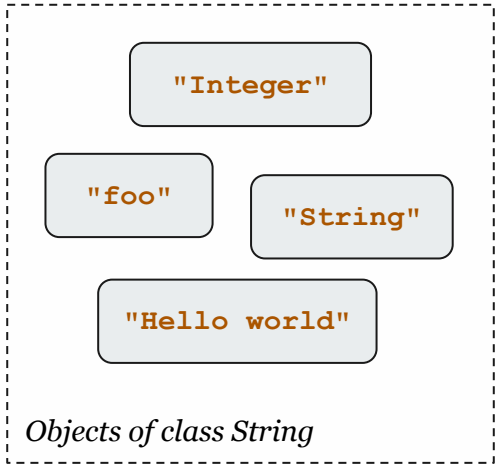
Every class is associated with a Class object...

And every object belongs to a class.

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

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

// This prints "String"
```



Objects of class String

class Class

In java.lang.Class

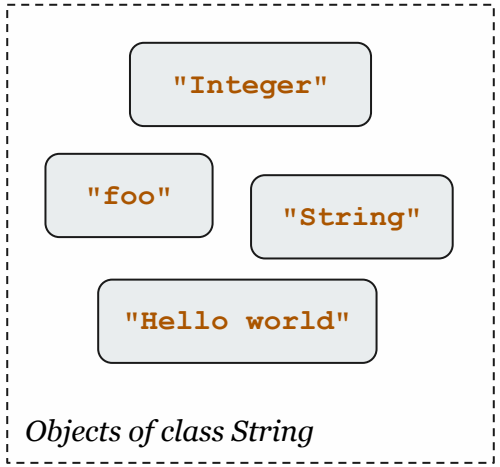
Every class is associated with a Class object...

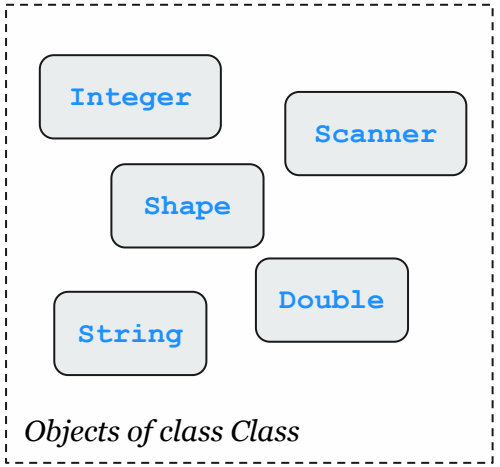
And every object belongs to a class.

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

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

// This prints "String"
```





class Class

In java.lang.Class

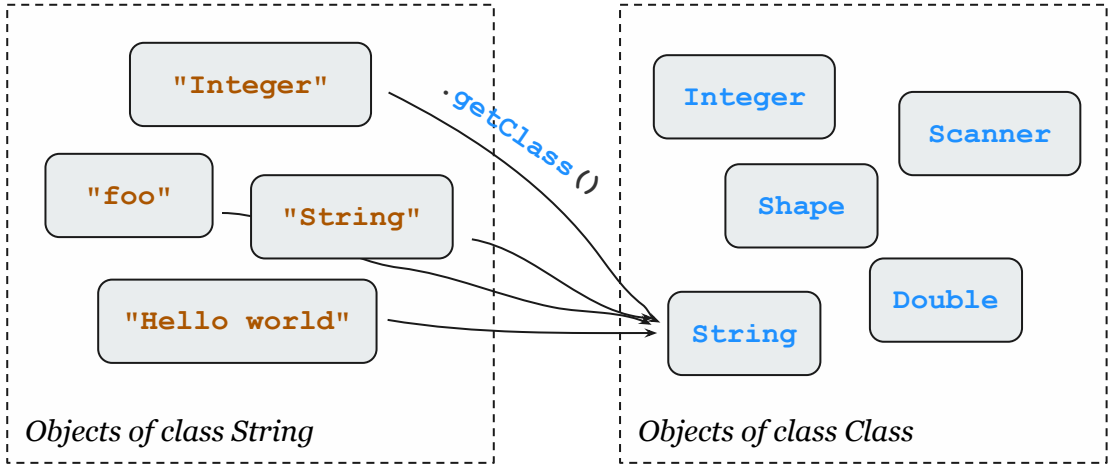
Every class is associated with a Class object...

And every object belongs to a class.

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

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

// This prints "String"
```



class Class

In java.lang.Class

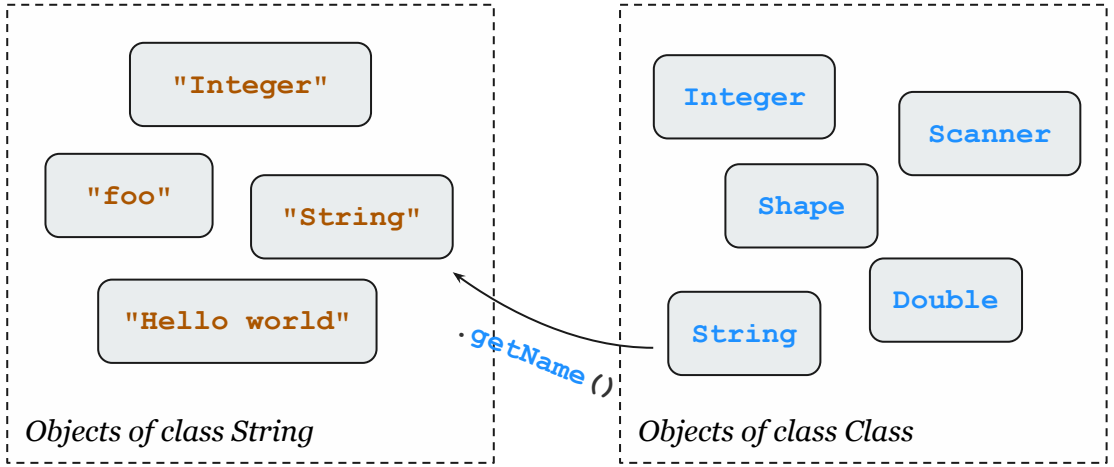
Every class is associated with a Class object...

And every object belongs to a class.

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

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

// This prints "String"
```



class Class

In java.lang.Class

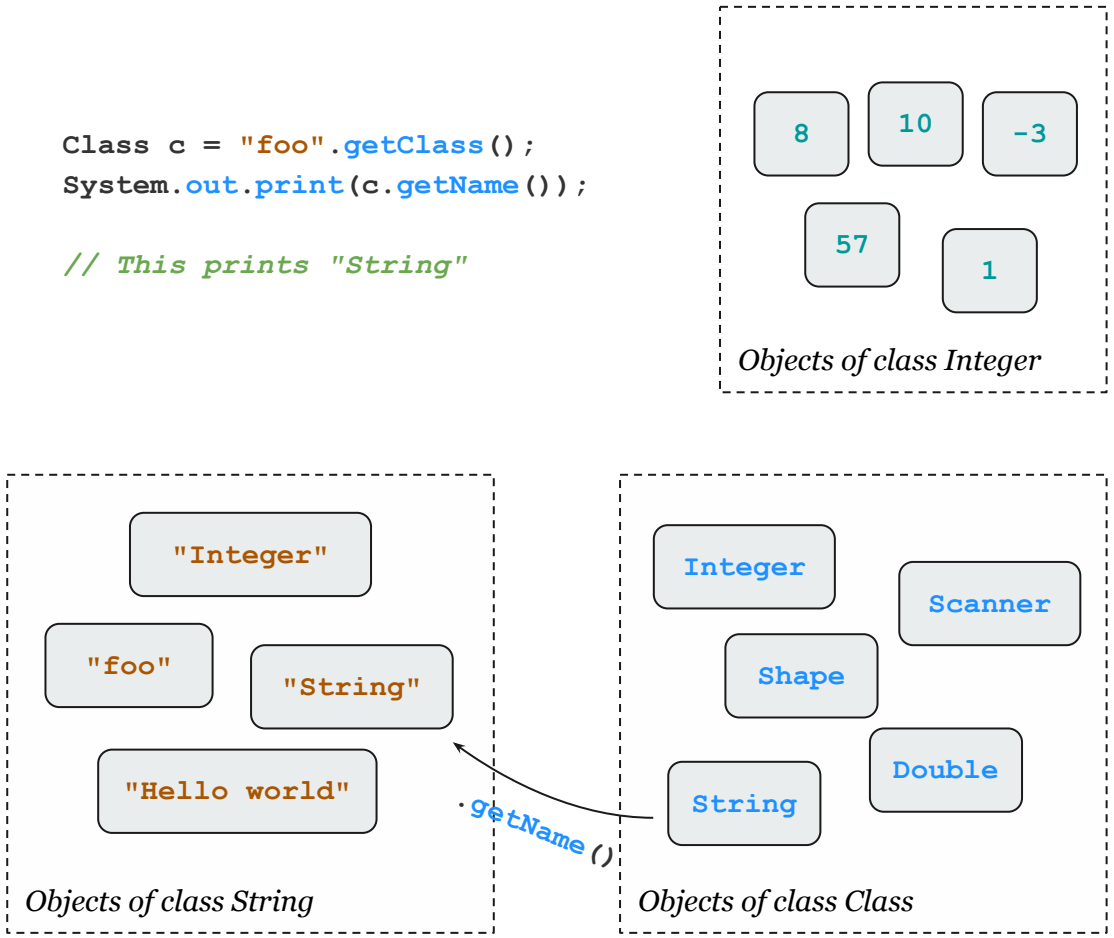
Every class is associated with a Class object...

And every object belongs to a class.

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

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

// This prints "String"
```



class Class

In java.lang.Class

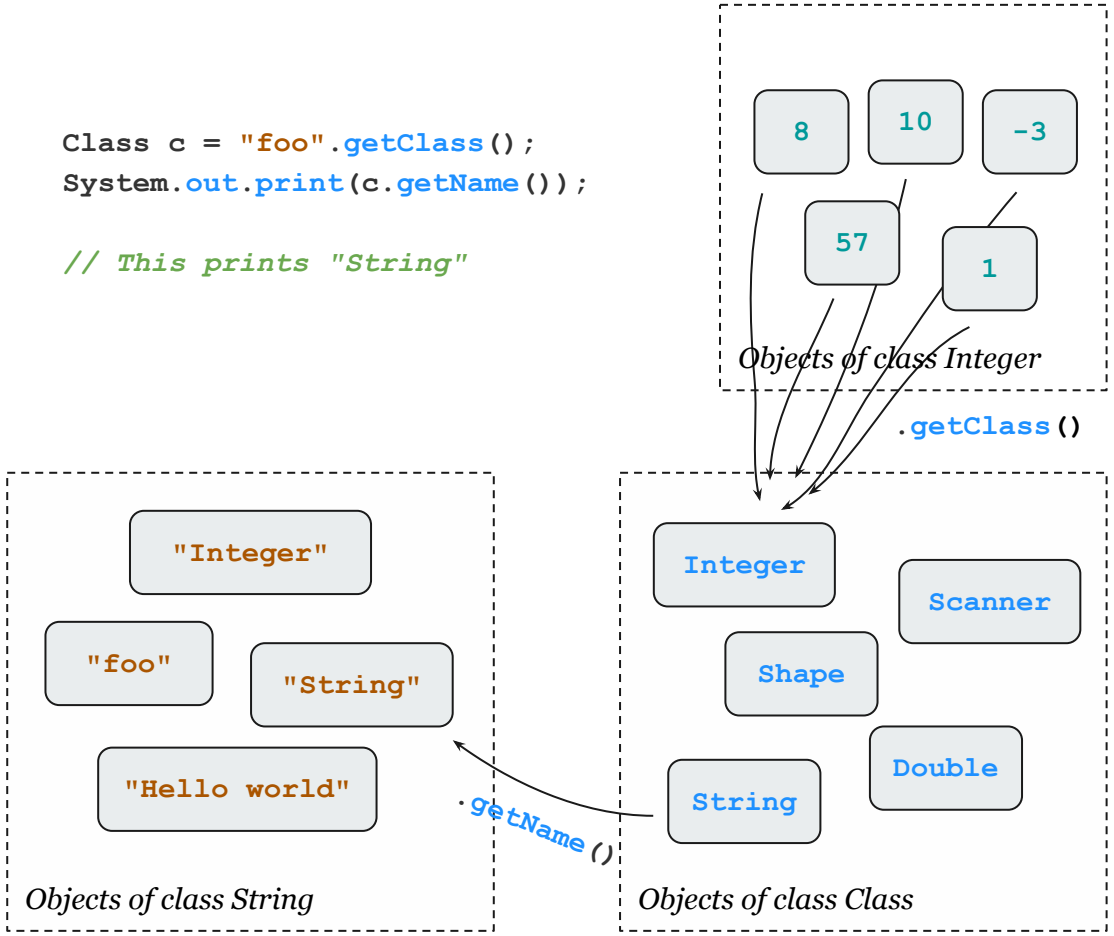
Every class is associated with a Class object...

And every object belongs to a class.

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

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

// This prints "String"
```



class Class

In java.lang.Class

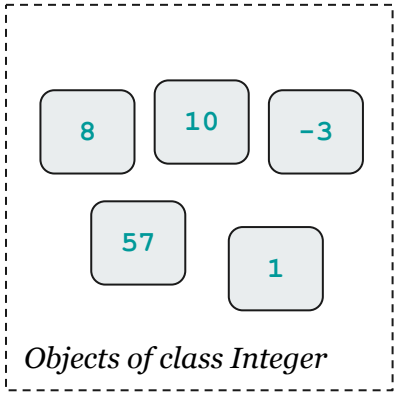
Every class is associated with a Class object...

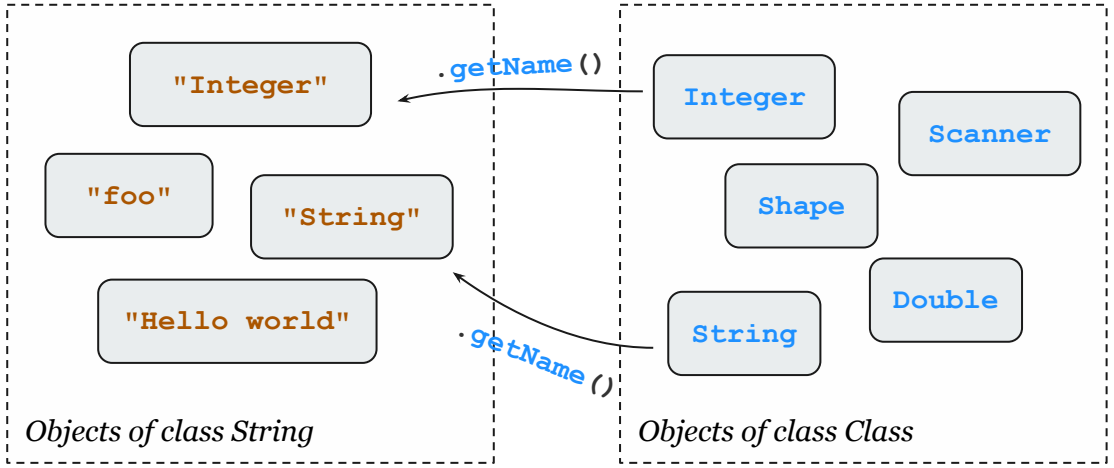
And every object belongs to a class.

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

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

// This prints "String"
```





class Class<T>

In java.lang.Class

`static Class<T>`	`forName(String name)`
`String`	`getName()`
`T`	`cast(Object obj)`
`boolean`	`isInstance(Object obj)`
`Class<?>[]`	`getClasses()`
`Field`	`getField(String name)`
`Constructor<T>`	`getConstructor( … )`
`Method`	`getMethod(String name, … )`
`Field[]`	`getFields()`
`Constructor<?>[]`	`getConstructors()`
`Method[]`	`getMethods()`

class Field

In the java.lang.reflect *package*

`String`	`getName()`
`Class`	`getType()`
`Object`	`get(Object obj)`
`int / double / …`	`getInt / getDouble / …`
`void`	`set(Object obj, Object value)`
`void`	`setInt / 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);
```

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 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 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*

`String`	`getName()`
`Class`	`getDeclaringClass()`
`int`	`getParameterCount()`
`Class<?>[]`	`getParameterTypes()`
`T`	`newInstance( … )`

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*

`String`	`getName()`
`Class`	`getDeclaringClass()`
`int`	`getParameterCount()`
`Class<?>[]`	`getParameterTypes()`
`Class<?>`	`getReturnType()`
`Object`	`invoke(Object obj, … )`

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??"

```
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??"

```
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??"

```
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()
```

```
t.getClass().getMethod("test").invoke(t);
```

Performance

The elephant in the room...

```
p.test()
```

≠

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

Go test it in **Eclipse**!


DEPARTMENT OF
COMPUTER SCIENCE
Thanks!
Any Questions?