INTRODUCTION TO JAVA

Java 1.0

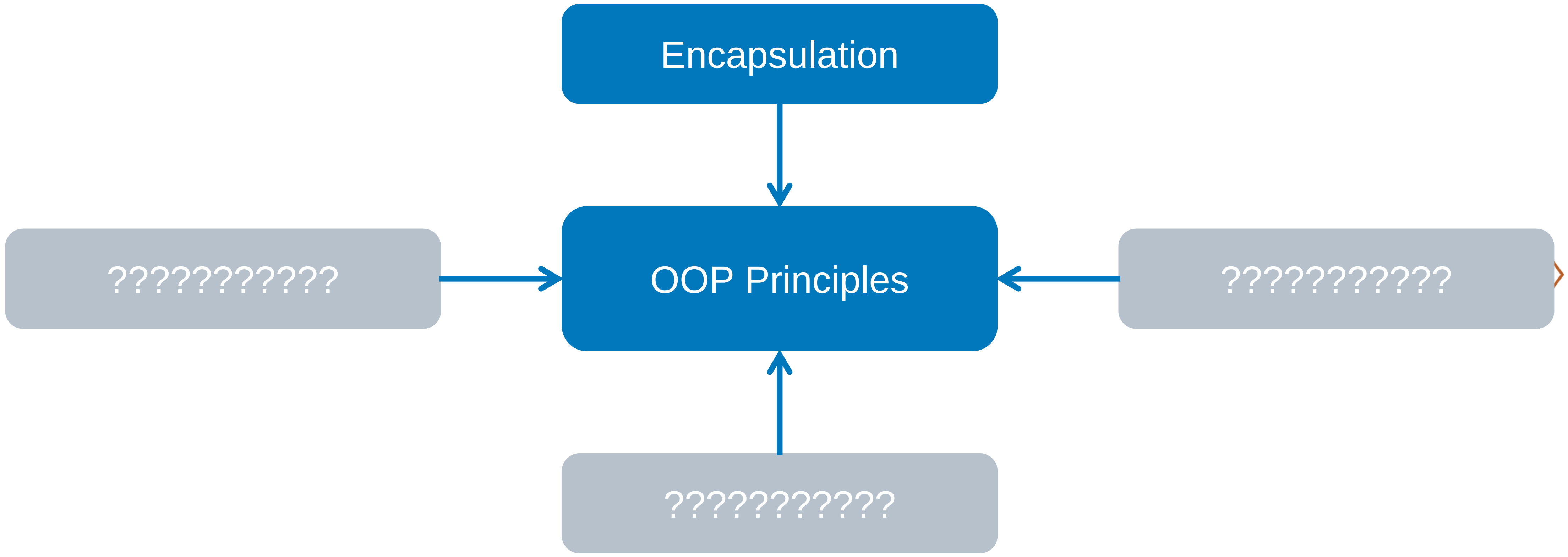
ENCAPSULATION

Lesson # 07

OBJECT-ORIENTED PROGRAMMING CONCEPTS

JavaGuru

PILLARS OF OBJECT-ORIENTED PROGRAMMING



ENCAPSULATION OVERVIEW

- Binding of data and behavior together in a single unit
- Data is not accessed directly, but through the methods present inside class
- Makes the concept of data hiding possible

ACCESS MODIFIERS OVERVIEW

- Specifies which classes can access a given class and its fields, constructors and methods
- Classes, fields, constructors and methods can have one of four different access modifiers:
    - private
    - default (package private)
    - protected
    - public

PRIVATE ACCESS MODIFIER

- When element is declared as private, then only code inside the same class can access it
- Declarable code elements:
    - Fields (variables)
    - Methods
    - Constructors
- Restricted code elements:
    - Classes

DEFAULT(PACKAGE PRIVATE) ACCESS MODIFIER

- When element is declared as package private, then only code inside the same class or within the same package can access it
- Declarable code elements:
  - Fields (variables)
  - Methods
  - Constructors
  - Classes

PUBLIC ACCESS MODIFIER

- When element is declared as public, then all code regardless of location can access it
- Declarable code elements:
  - Fields (variables)
  - Methods
  - Constructors
  - Classes

BASIC COUNTER - REQUIREMENTS

- State
  - Current counter value cannot be accessed directly
- Behavior
  - Can increment, decrement and clear counter value
  - Can set counter value to any specified positive number (otherwise set to 0)
  - Can be constructed only within the same package

BASIC COUNTER – NO DIRECT ACCESS

Hide external state of counter by marking it as private

Allow external access by providing getter method

```
public class BasicCounter {

    private int counter;

    public int getCounter() {
        return counter;
    }

}
```

BASIC COUNTER – PRIMARY BEHAVIOR

Control counter from outside without access to its state

```
public class BasicCounter {
    ...

    public void increment() {
        counter++;
    }

    public void decrement() {
        counter--;
    }

    public void clear() {
        counter = 0;
    }

}
```

BASIC COUNTER – SECONDARY BEHAVIOR

```java
public class BasicCounter {

    ...

    public void setCounter(int counter) {
        if (isPositive(counter)) {
            this.counter = counter;
        } else {
            clear();
        }
    }

    private boolean isPositive(int value) {
        return value > 0;
    }

}
```

Only counter knows about validation rules

BASIC COUNTER – CONSTRUCTION LIMITATIONS

No access modifier specified means it can be called only within the same package

```
public class BasicCounter {

    ...

    BasicCounter() {

    }

    ...

}
```

Empty constructor

BASIC COUNTER – FINAL RESULT

```java
public class BasicCounter {
    private int counter;

    BasicCounter() {
    }

    public int getCounter() {
        return counter;
    }

    public void setCounter(int counter) {
        if (isPositive(counter)) {
            this.counter = counter;
        } else {
            clear();
        }
    }

    public void increment() {
        counter++;
    }

    public void decrement() {
        counter--;
    }

    public void clear() {
        counter = 0;
    }

    private boolean isPositive(int value) {
        return value > 0;
    }
}
```

{JG} JavaGuru

OBJECT EQUALITY AND IDENTITY

OBJECT AND HEAP MEMORY REVISION

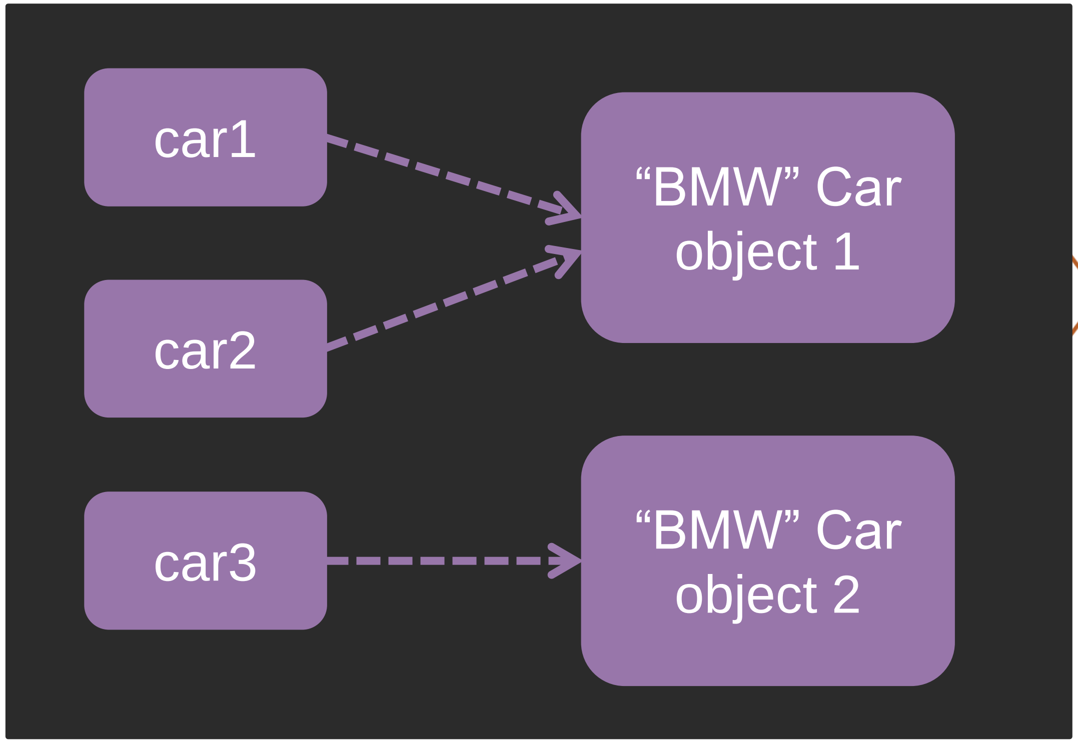
- When object is created, it is being stored in the heap memory
- To be able to locate an object, computer assigns it an address in the memory
- Every new object created gets a new address

OBJECT AND HEAP MEMORY REVISION

Code view

```
Car car1 = new Car("BMW");
Car car2 = car1;
Car car3 = new Car("BMW");
```

Objects in memory view



REFERENCE EQUALITY RELATIONAL OPERATOR

- Relational operator == used to compare two operands and determine whether the two operands are equal or not
- When used on referential type, we can see if both variables refer to the same object in the heap memory

REFERENCE EQUALITY – CODE EXAMPLE

```java
Car car1 = new Car("BMW");
Car car2 = car1;
Car car3 = new Car("BMW");

if (car1 == car1) { //true
}

if (car1 == car2) { //true
}

if (car1 == car3) { //false
}
```

LOGICAL EQUALITY – METHOD EQUALS

- Every class by default has equals method that compares object method was called on with specified parameter
- Compares the data of the objects instead of the value of the references

LOGICAL EQUALITY – CODE EXAMPLE

```
Car car1 = new Car("BMW");
Car car2 = car1;
Car car3 = new Car("BMW");

if (car1.equals(car1)) { //true
}

if (car1.equals(car2)) { //true
}

if (car1.equals(car3)) { //false
}
```

SAME, BUT DIFFERENT, BUT STILL SAME

Object class default equals method implementation

By default it compares by reference

```java
public boolean equals(Object obj) {
    return (this == obj);
}
```

OVERRIDE DEFAULT BEHAVIOR

- Default method implementation knows nothing about concrete class data, hence reference comparison by default
- Control what data of the class should be compared and how it should be done

{JG} JavaGuru

OVERRIDE DEFAULT BEHAVIOR EXAMPLE

```java
public class Car {

    private String make;

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Car car = (Car) obj;
        return Objects.equals(make, car.make);
    }

    public Car(String make) {
        this.make = make;
    }

}
```

Check if both reference the same object

Check argument is not null and both are instances of the same class

Cast argument to the Car type

Specify which class fields to compare

LOGICAL EQUALITY – OVERRIDE CODE EXAMPLE

```
Car car1 = new Car("BMW");
Car car2 = car1;
Car car3 = new Car("BMW");

if (car1.equals(car1)) { //true
}

if (car1.equals(car2)) { //true
}

if (car1.equals(car3)) { //true
}
```

STRING INSTANTIATION

Instantiating String object without new keyword

```java
String artist = "Taylor Swift";

String band = new String("Metallica");
```

Instantiating String object with new keyword

EQUALITY DIFFERENCE

Reference Equality

```
String cat1 = "Cat";
String cat2 = "Cat";
String cat3 = new String("Cat");

if (cat1 == cat2) { //true
}

if (cat1 == cat3) { //false
}
```

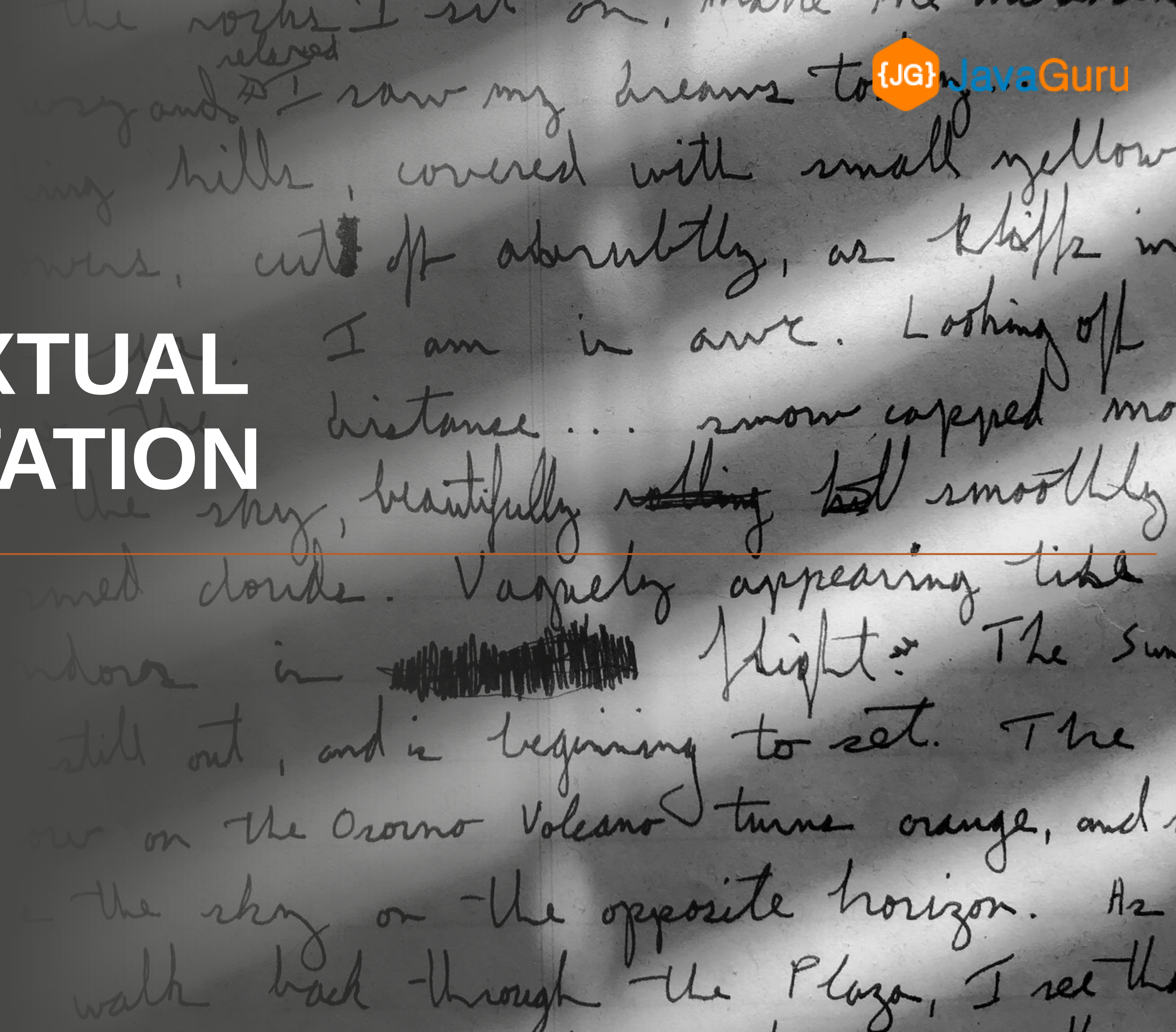
Logical equality

```
String cat1 = "Cat";
String cat2 = "Cat";
String cat3 = new String("Cat");

if (cat1.equals(cat2)) { //true
}

if (cat1.equals(cat3)) { //true
}
```

OBJECT TEXTUAL REPRESENTATION



WRITING OBJECT DETAILS TO THE CONSOLE

```
public class SmartPhone {

    private String manufacturer;
    private String model;

    public SmartPhone(String manufacturer, String model) {
        this.manufacturer = manufacturer;
        this.model = model;
    }

    public String getManufacturer() {
        return manufacturer;
    }

    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }
}
```

WRITING OBJECT DETAILS TO THE CONSOLE

Code

```java
SmartPhone phone = new SmartPhone("Apple", "iPhone 14 Pro");

System.out.println("Manufacturer: " + phone.getManufacturer());
System.out.println("Model: " + phone.getModel());
```

Console output

```
Brand: Apple
Model: iPhone 14 Pro
```

WRITING OBJECT DETAILS TO THE CONSOLE

Code

```
SmartPhone phone = new SmartPhone("Apple", "iPhone 14 Pro");

System.out.println(phone);
```

Console output

```
lv.javaguru.training.lesson7.SmartPhone@1b28cdfa
```

DEFAULT TO STRING METHOD

Start with declared class name

Separated with @ symbol

```java
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

End with HEX representation of integer has of the object

OVERRIDE TO STRING METHOD

```
public class SmartPhone {

    private String manufacturer;
    private String model;
    ...
    @Override
    public String toString() {
        return "SmartPhone{" +
                "manufacturer='" + manufacturer + '\'' +
                ", model='" + model + '\'' +
                '}';
    }

}
```

WRITING OBJECT DETAILS TO THE CONSOLE

Code

```java
SmartPhone phone = new SmartPhone("Apple", "iPhone 14 Pro");

System.out.println(phone);
```

Console output

```
SmartPhone{manufacturer='Apple', model='iPhone 14 Pro’}
```

REFERENCES

REFERENCES

- https://dzone.com/articles/object-identity-and-equality-injava
- https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#toString--
- https://users.soe.ucsc.edu/~eaugusti/archive/102-winter16/misc/howToOverrideEquals.html

JavaGuru

QUESTIONS?

{JG} JavaGuru

THANK YOU!