



Java Training OOP

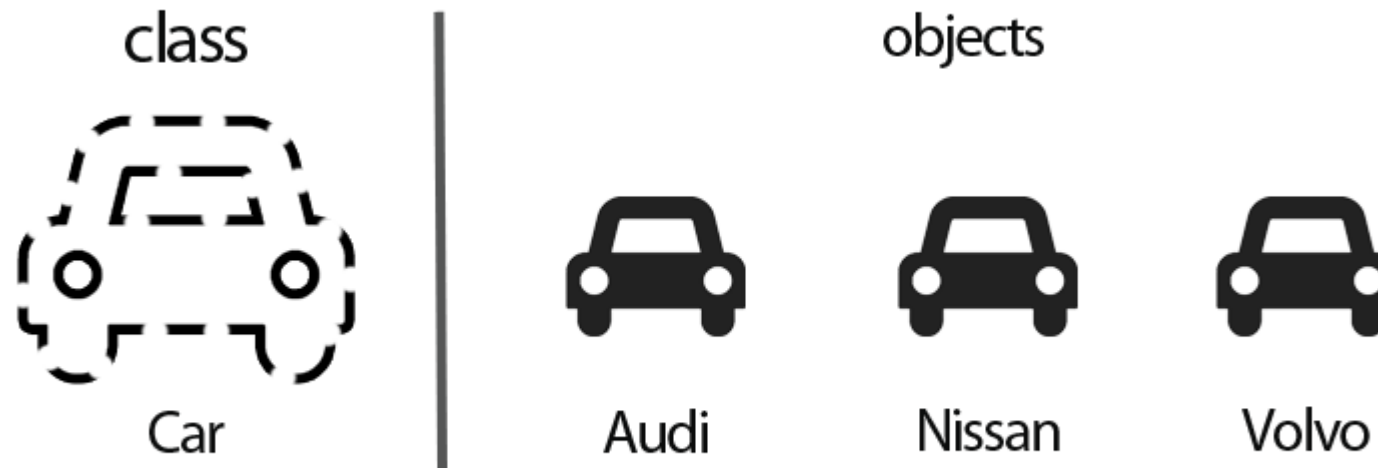
Agenda

- ❖ Classes and Objects
- ❖ Constructors
- ❖ Enum
- ❖ Encapsulation, Inheritance, Polymorphism
- ❖ Static methods
- ❖ JavaBeans Fundamentals



Classes and Objects

- **Object** – a specified entity in our program
- **Class** – a template or abstraction of our program entities , like a blueprint



<https://javatutorial.net/java-oop>



Entity Class

```
public class Car {  
    private double fuel;  
    private double currentSpeed;  
  
    public Car() {}  
  
    public Car(double fuel, double currentSpeed) {  
        this.currentSpeed = currentSpeed;  
        this.fuel = fuel;  
    }  
  
    public double getFuel() {  
        return fuel;  
    }  
  
    public void setFuel(double fuel) {  
        this.fuel = fuel;  
    }  
    ...  
}
```



Objects

```
public static void main(String[] args) {  
    Car sportCar = new Car(50, 250);  
    Car van = new Car(100, 120);  
}
```

Objects

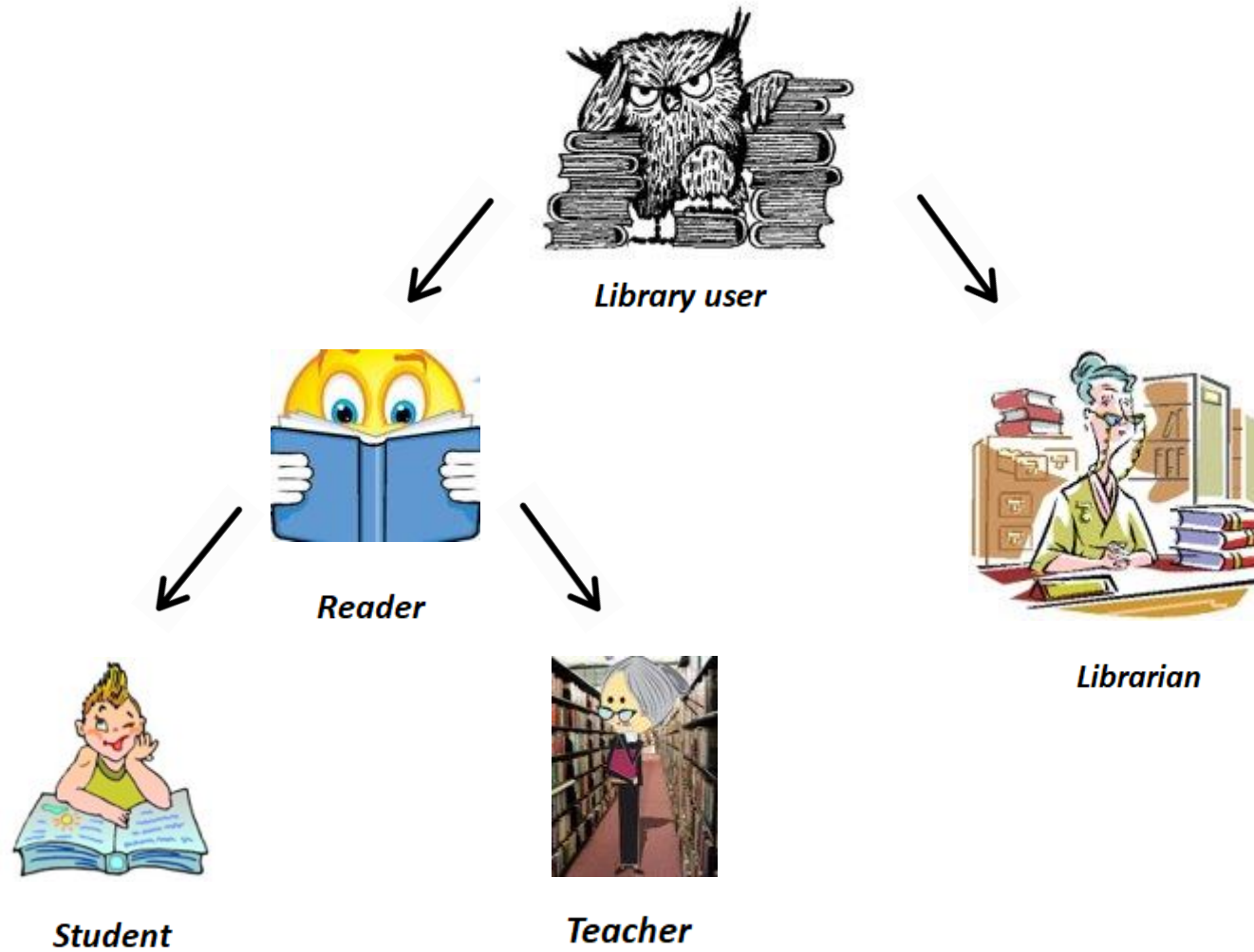


Class

<https://csawesome.runestone.academy/runestone/books/published/csawesome/Unit2-Using-Objects/topic-2-1-objects-intro-turtles.html>



Inheritance



Inheritance

- Inheritance is the mechanism by which an object acquires the some/all properties of another object.
- It supports the concept of hierarchical classification.

```
public class Van extends Car {  
    private double loadCapacity;  
  
    public Van(double fuel, double currentSpeed, ColorEnum color, double loadCapacity) {  
        super(fuel, currentSpeed, color);  
        this.loadCapacity = loadCapacity;  
    }  
  
    public double getLoadCapacity() {  
        return loadCapacity;  
    }  
  
    public void setLoadCapacity(double loadCapacity) {  
        this.loadCapacity = loadCapacity;  
    }  
}
```



Immutable Objects

- Object which state can't be modified after the creation

```
public final class ImmutableCar {  
    private final Car car;  
  
    public ImmutableCar(Car car) {  
        this.car = car;  
    }  
  
    public Car setCar(Car car) {  
        return new ImmutableCar(car);  
    }  
}
```



Enum

```
public enum Season {  
    SPRING,  
    SUMMER,  
    AUTUMN,  
    WINTER  
}
```



<https://proft.me/2015/03/29/primer-raboty-s-perechisleniyami-enum-v-java/>

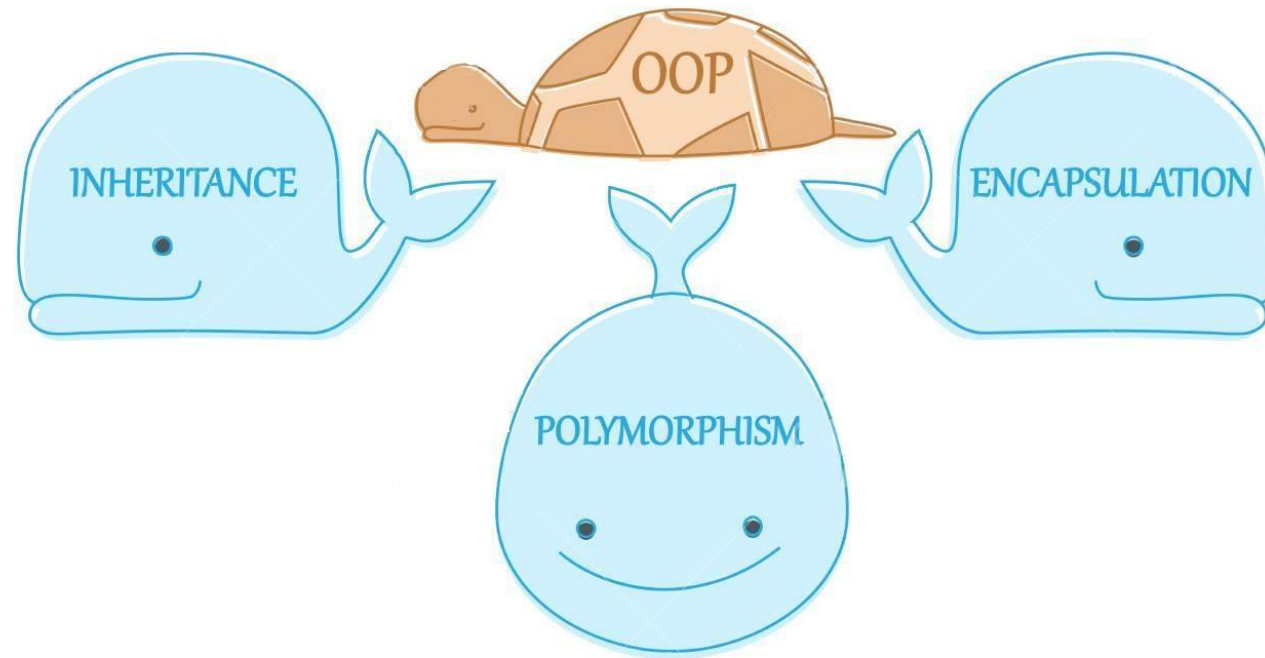


Old style enum

```
public class Season {  
    public static final Season SPRING = new Season("SPRING");  
    public static final Season SUMMER = new Season("SUMMER");  
    public static final Season AUTUMN = new Season("AUTUMN");  
    public static final Season WINTER = new Season("WINTER");  
  
    private String seasonName;  
  
    private Season(String seasonName) {  
        this.seasonName = seasonName;  
    }  
}
```



Encapsulation, Inheritance, Polymorphism



Encapsulation



private

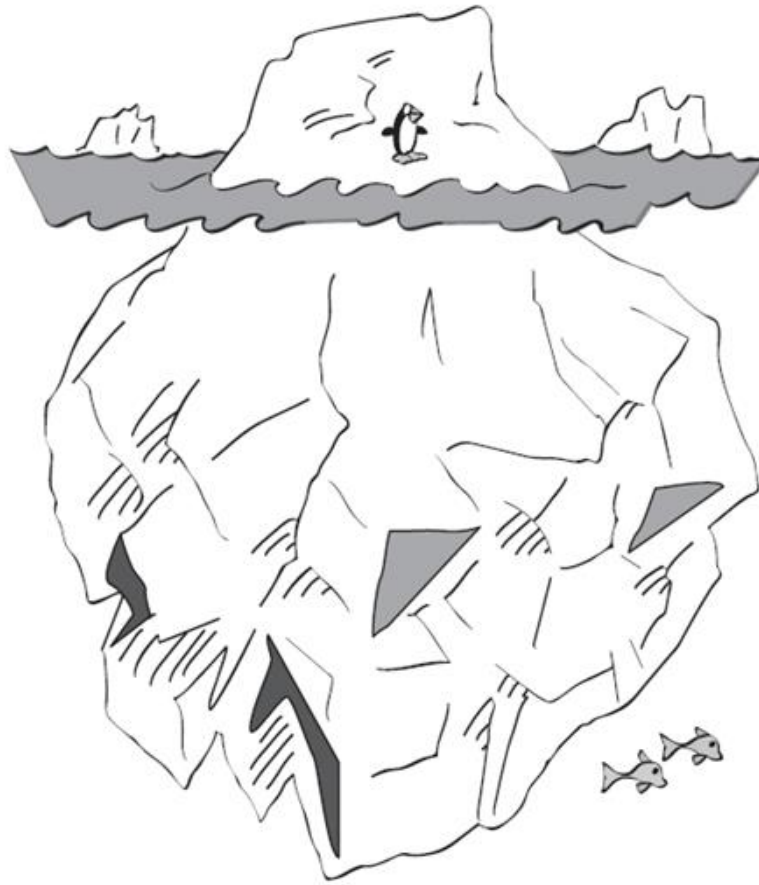
Package private

Protected

Public



Encapsulation



Encapsulation

- Binding the data with the code that manipulates it.
- It keeps the data and the code safe from external interference

```
public class Word {  
    private chars[] letters;  
  
    public Word(char[] letters) {  
        this.letters = letters;  
    }  
  
    public Word(String letters) {  
        this.letters = letters.toCharArray();  
    }  
  
    public void setLetters(chars[] letters) {  
        this.letters = letters;  
    }  
  
    public void setLetters(String letters) {  
        this.letters = letters.toArray();  
    }  
}
```



Polymorphism

- Polymorphism means to process objects differently based on their data type.
- In other words it means, one method with multiple implementation, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object)



Polymorphism: Overriding

```
public interface Calculator {  
    double calculate(double firstNumber, double secondNumber);  
}
```



```
public class MultiplicationCalculator implements Calculator{  
    @Override  
    public double calculate(double firstNumber, double secondNumber) {  
        return firstNumber * secondNumber;  
    }  
}
```



```
public class SumCalculator implements Calculator {  
    @Override  
    public double calculate(double firstNumber, double secondNumber) {  
        return firstNumber + secondNumber;  
    }  
}
```



Polymorphism: Overriding

```
public class Service {
    private Calculator calculator;

    public Service(Calculator calculator) {
        this.calculator = calculator;
    }

    public double calculate(double firstNumber, double secondNumber) {
        return calculator.calculate(firstNumber, secondNumber);
    }
}

public static void main(String[] args) {
    Calculator multiplicationCalculator = new MultiplicationCalculator();
    Calculator sumCalculator = new SumCalculator();
    Service sumService = new Service(multiplicationCalculator);
    Service multiplicationService = new Service(sumCalculator);

    double firstNumber = 4;
    double secondNumber = 5;
    System.out.println(sumService.calculate(firstNumber, secondNumber));
    System.out.println(multiplicationService.calculate(firstNumber, secondNumber));
}
```

```
C:\dev_tools\jdk1.8.0_282\bin\java.exe ...
9.0
20.0

Process finished with exit code 0
```

Overloading

```
public class Car {  
    private double fuel;  
    private double currentSpeed;  
    private Color color;  
  
    public void refuel(double fuel) {  
        this.fuel += fuel;  
    }  
  
    public void refuel() {  
        this.fuel += 50;  
    }  
}
```

```
public static void main(String[] args) {  
    Car car = new Car(100, 100, Color.BLUE);  
    System.out.println("Fuel: - " + car.getFuel());  
  
    car.refuel();  
    System.out.println("Fuel: - " + car.getFuel());  
  
    car.refuel(10);  
    System.out.println("Fuel: - " + car.getFuel());  
}
```

```
C:\dev_tools\jdk1.8.0_282\bin\java.exe ...  
Fuel: - 100.0  
Fuel: - 150.0  
Fuel: - 160.0  
  
Process finished with exit code 0
```



Static methods

- ❖ Method is attached to the class itself not to specific object
- ❖ There is no *this* and *super* link
- ❖ Early binding
- ❖ Can't interact with non-static blocks
- ❖ Can't be properly overridden



Static methods

```
public class CarService {  
    private static final double FUEL_PER_FILLING = 50;  
  
    public static void fillUpCar(Car car) {  
        car.setFuel(car.getFuel + FUEL_PER_FILLING);  
    }  
}  
  
public static void main(String[] args) {  
    Car sportCar = new Car(150, 100);  
    CarService.fillUpCar(sportCar);  
}
```



Static initialization blocks

Used to initialize static resources

```
public class StaticClass {  
    private static final String CONSTANT_VALUE;  
    private static Integer counter;  
  
    static {  
        CONSTANT_VALUE = "Initialized in the static block";  
        counter = 0;  
    }  
  
}
```



Class Object

```
public class Object {  
    public Object() {  
    }  
  
    private static native void registerNatives();  
  
    public final native Class<?> getClass();  
  
    public native int hashCode();  
  
    public boolean equals(Object var1) {  
        return this == var1;  
    }  
  
    protected native Object clone() throws CloneNotSupportedException;  
  
    public String toString() {  
        return this.getClass().getName() + "@" + Integer.toHexString(this.hashCode());  
    }  
  
    ...  
}
```



Equals + Hashcode

@Override

```
public boolean equals(Object o) {  
    if (this == o) {  
        return true;  
    }  
    if (o == null || getClass() != o.getClass()) {  
        return false;  
    }  
    Car car = (Car) o;  
    return Double.compare(car.fuel, fuel) == 0 &&  
        Double.compare(car.currentSpeed, currentSpeed) == 0;  
}
```

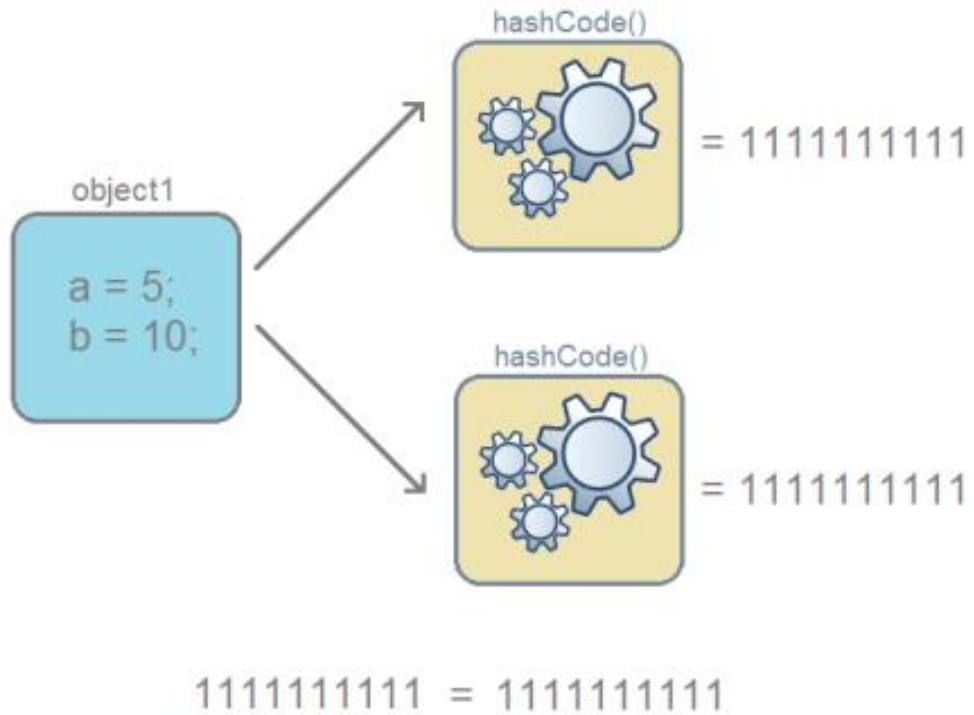
@Override

```
public int hashCode() {  
    return Objects.hash(fuel, currentSpeed);  
}
```

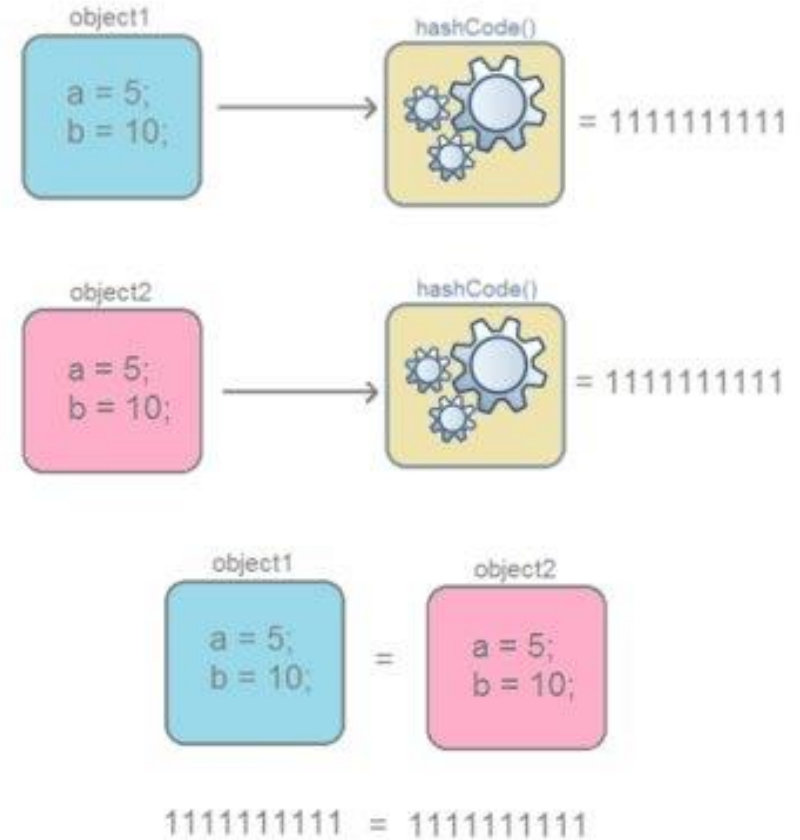


Equals Hashcode contract

Same object

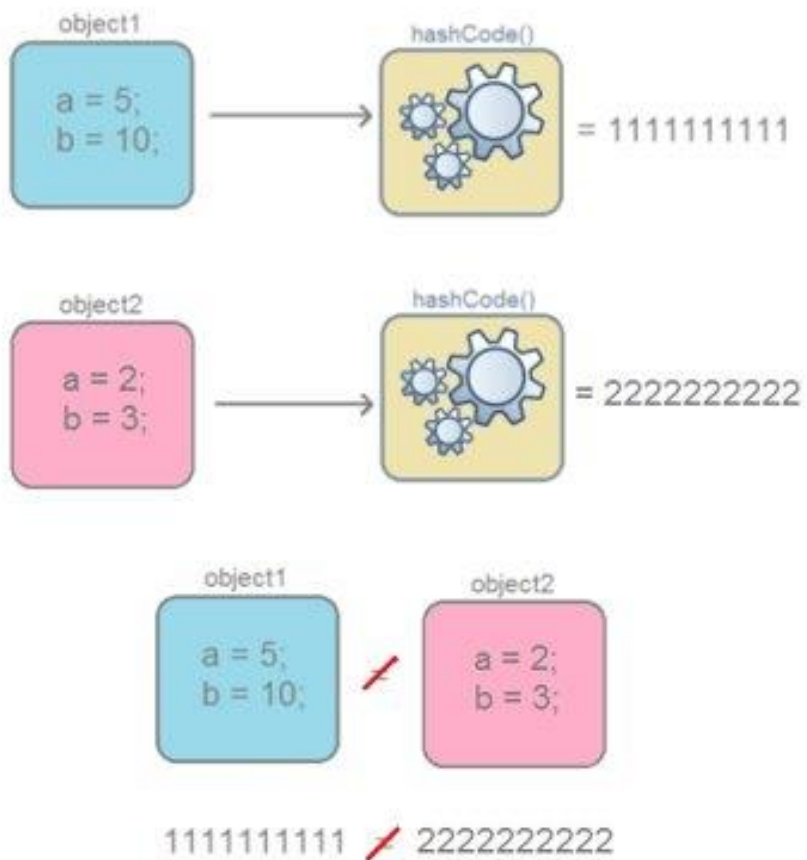


Similar objects

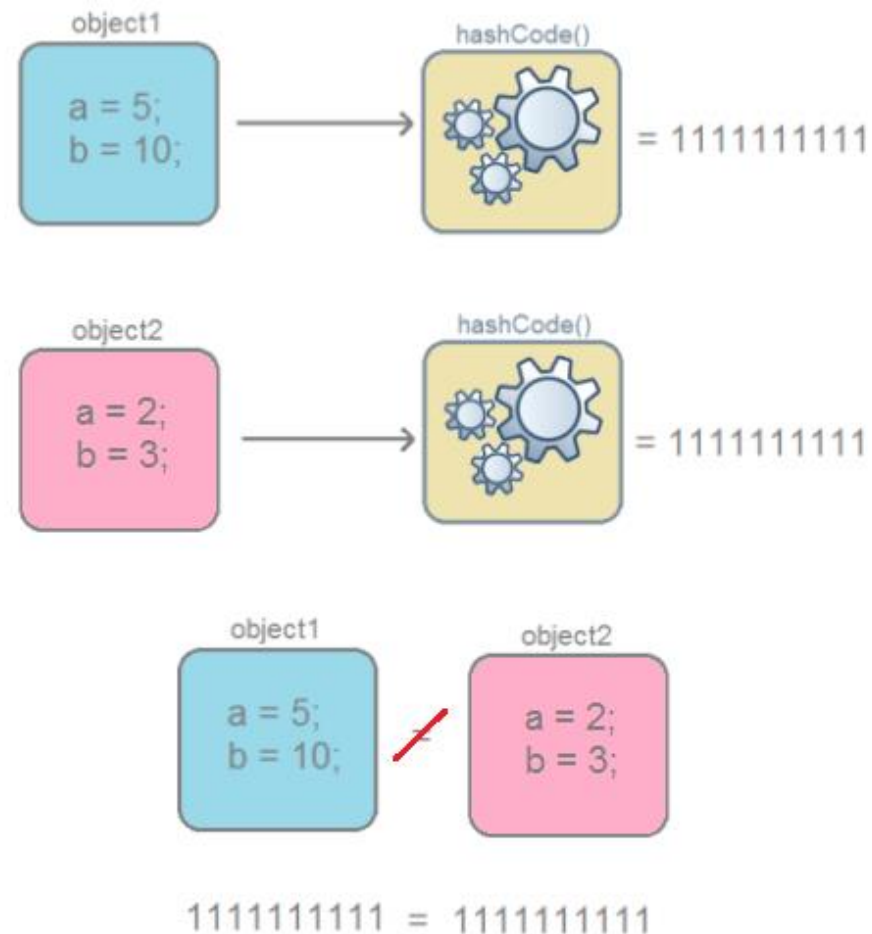


Equals Hashcode contract

Different objects



Different objects (collision)



Overriding Equals: Rules

- Reflexive: **x equals x**, an object must equal to itself.
- Symmetry: if **x equals y** then **y equals x == true**.
- Transitive: if **x equals y** and **y equals z** then **x equals z**
- Consistent: if **x equals y** and no value is modified, then it's always true for every call
- For any non-null object **x**, **x equals null == false**

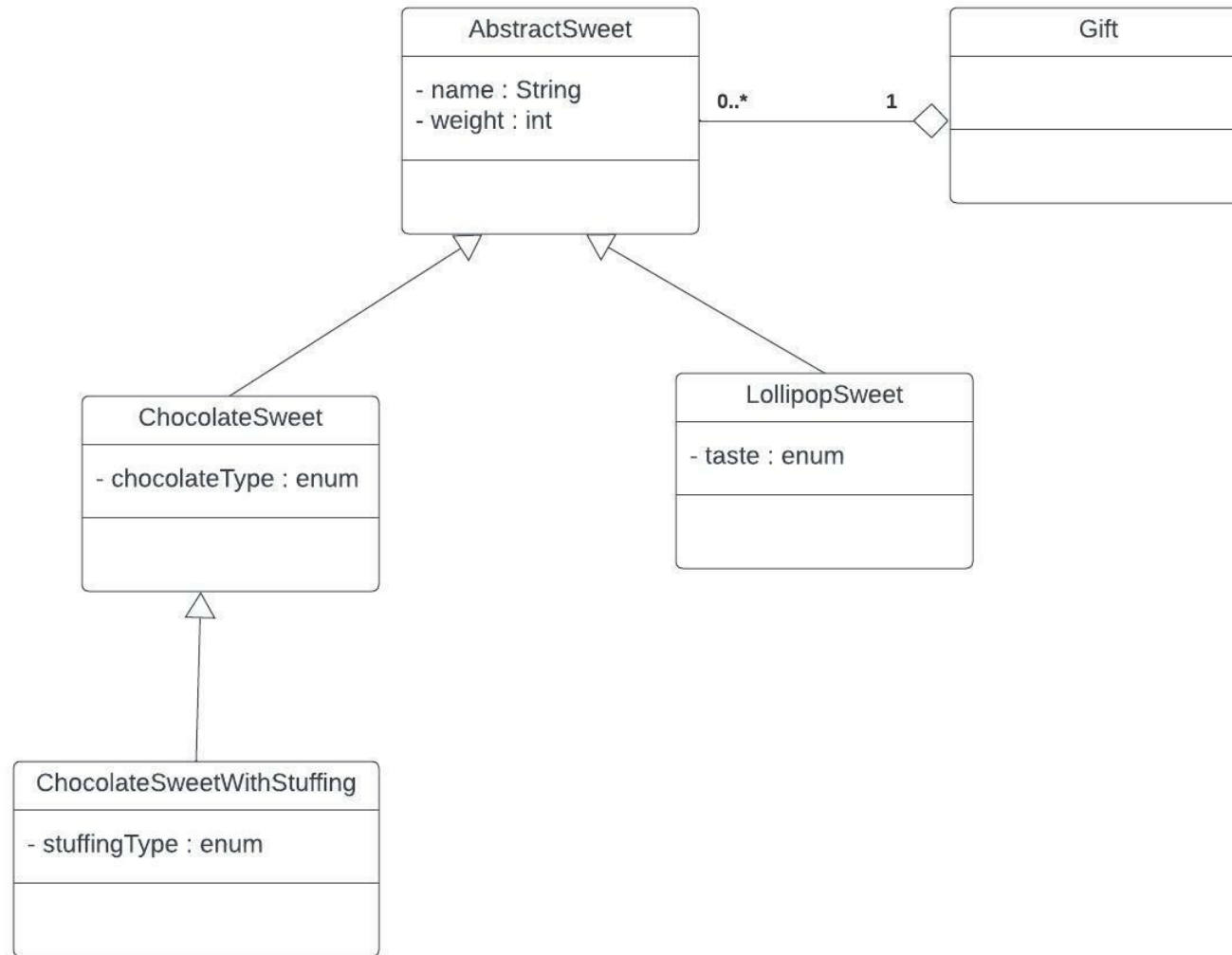


Homework

- **Christmas gift.** Define the hierarchy of candies and other sweets. Create several candy objects. Collect a child's gift and determine its weight.



Homework



Tasks

- 1. **Florist**. Define the hierarchy of flowers. Create several flower objects. Collect a bouquet and determine its value.
- 2. **Christmas gift**. Define the hierarchy of candies and other sweets. Create several candy objects. Collect a child's gift and determine its weight.
- 3. **Electrician**. Define the hierarchy of electrical appliances. Plug some of them into the socket. Calculate the power consumption.
- 4. **Diet**. Define the hierarchy of vegetables. Collect them into salad. Calculate calories.
- 5. **Music lover**. Define the hierarchy of musical tracks. Write the tracks to disk. Calculate the duration.
- 6. **Stones**. Define the hierarchy of precious and semi-precious stones. Select stones for the necklace. Calculate total weight (in carats) and value.
- 7. **Weapon**. Define the hierarchy of weapons (steel and firearms). Arming a military division. Calculate the value.
- 8. **Transport**. Define the hierarchy of passenger transport. Create a set of routes for moving from point A to point B. Calculate the total cost of travel.
- 9. **Airline**. Define the aircraft hierarchy. Create an airline. Calculate the total capacity (in passengers).
- 10. **Cars**. Define the hierarchy of passenger cars. Create a car company. Calculate the value of company's cars.



Books

- Sierra, Kathy. Head First Java
- Eckel, Bruce. Thinking in Java



Links

- <https://www.baeldung.com/java-classes-objects>
- <https://www.baeldung.com/java-constructors>
- <https://www.baeldung.com/a-guide-to-java-enums>
- https://www.w3schools.com/java/java_inheritance.asp
- <https://www.baeldung.com/java-method-overload-override>
- <https://dzone.com/articles/working-with-hashcode-and-equals-in-java>
- https://www.w3schools.com/java/java_encapsulation.asp
- <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>





Thanks