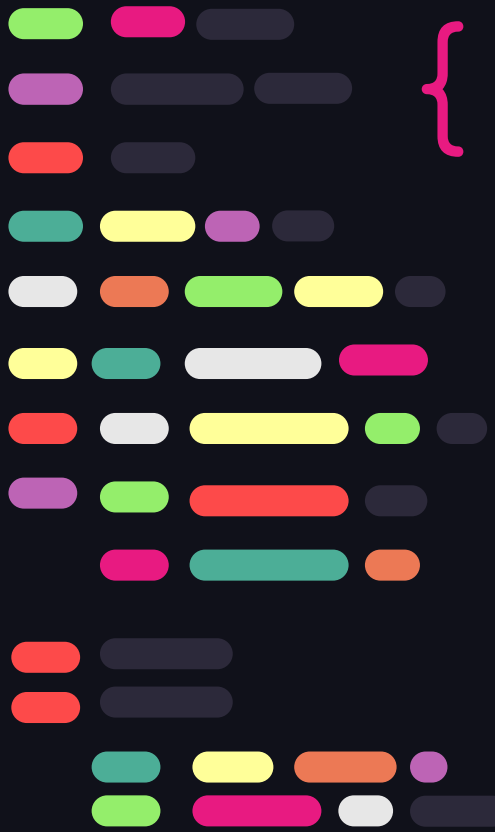
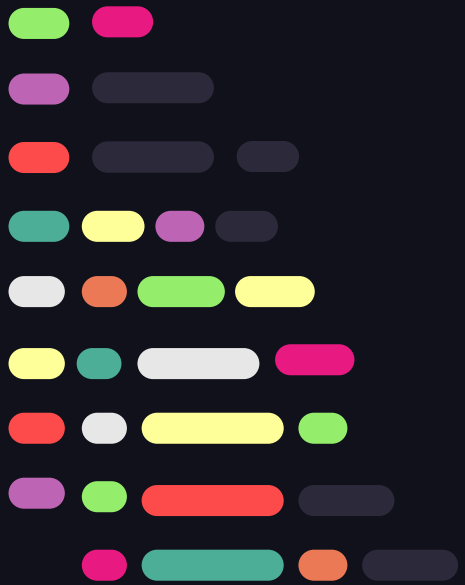


Expert OOP & Interface

Presentation By Aria Shakoo
Instructor : Dr. Mojtaba Vahidi Asl
1403 Ap Fall

Reviewing OOP



01 Class Attributions

Variables that belong to the class itself.

02 Class Methods

Methods are functions defined within a class

03 Getter & Setters

Methods used to access and modify the private attributes

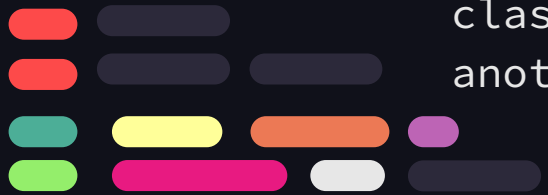
```java
public class Nature{
    private String Name;
    private int Age;
    private double Height;
    public Nature(String Name, int Age , double Height){
        this.Name = Name;
        this.Age = Age;
        this.Height = Height;
    }
    public String getName(){
        return Name;
    }
    public int getAge(){
        return Age;
    }
    public double Height(){
        return Height;
    }
    public void setAge(int Age ){
        this.Age = Age;
    }
    public void setName(String Name ){
        this.Name = Name;
    }
    public void setHeight(double Height){
        this.Height = Height;
    }
}
```

Example!}

01 {..

Inheritance

- Is a fundamental concept in OOP that allows a class to inherit attributes and methods from another class.

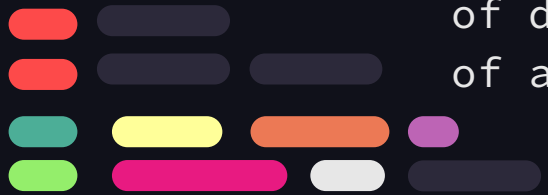


} ..

02 { ..

Polymorphism

- Is a core concept in OOP that allows objects of different classes to be treated as objects of a common superclass.



} . .

```java
class Tree extends Nature{
    public Tree (String Name , int Age , double Height){
        super(Name, Age, Height);
    }

    @Override
    public void Speak(){
        System.out.println("I am a Tree");
    }
}

class Bush extends Nature{
    public Bush (String Name , int Age , double Height ){
        super(Name , Age , Height);
    }

    @Override
    public void Speak(){
        System.out.println("I am a Bush");
    }
}
```

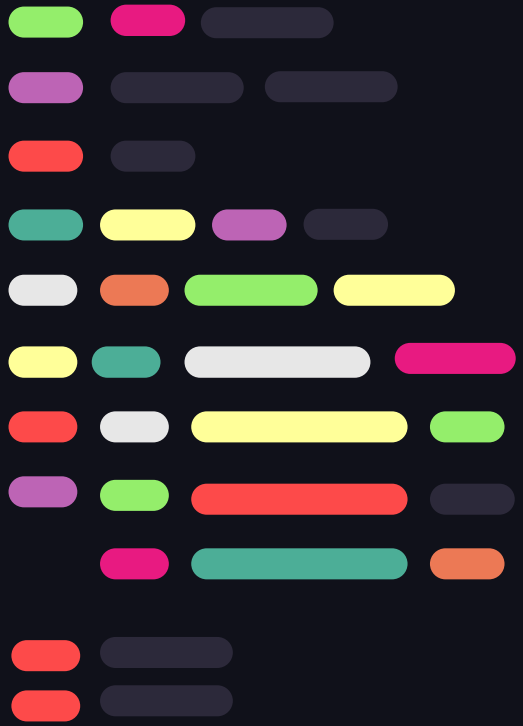
Example!}

```java
public class Main{
    public static void main(String[] args) {
        Nature tree = new Tree("Oak" , 100 , 30.5);
        Nature bush = new Bush("Rose" , 2 , 1.2);

        tree.Speak(); // I am a Tree
        bush.Speak(); // I am a Bush
    }
}
```

Example!}

*

Important Methods for Exam!

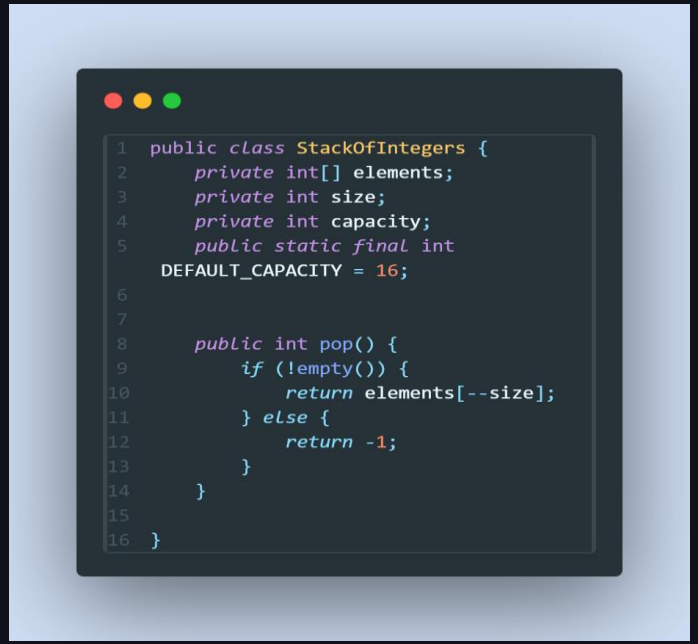


- Add
- Pop
- Push

Add Method

```java
public class Course {
    private String name;
    private String[] students;
    private int numOfStudents;
    int capacity;

    // increase the array size

    public void addStudents(String student) {
        if (numOfStudents < capacity) {
            students[numOfStudents++] = student;
        } else {
            capacity *= 2;
            String[] temp = new String[capacity];
            System.arraycopy(students, 0, temp, 0, students.
length);
            students = temp;
            students[numOfStudents++] = student;
        }
    }

}
```

Pop Method



```java
public class StackOfIntegers {
    private int[] elements;
    private int size;
    private int capacity;
    public static final int
 DEFAULT_CAPACITY = 16;


    public int pop() {
        if (!empty()) {
            return elements[--size];
        } else {
            return -1;
        }
    }

}
```

Push Method

```java
public class StackOfIntegers {
    private int[] elements;
    private int size;
    private int capacity;
    public static final int DEFAULT_CAPACITY
= 16;

    public int push(int value) {
        if (size < capacity) {
            elements[size++] = value;
            return 1;
        } else {
            capacity *= 2;
            int[] temp = new int[capacity];
            System.arraycopy(elements, 0,
 temp, 0, elements.length);
            elements = temp;
            elements[size++] = value;
            return 1;
        }
    }
}

```

Here are the three Methods

Pop

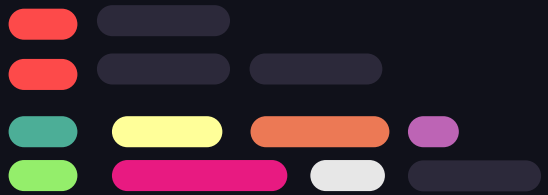
- Check if it is empty or no and don’t forget to re arrange the array again

Add

- Remember to increase the size of the array and re arrange the array

Push

Same as the add method but remember to return a value that shows it was successfull



Learning new things

Abstract classes

- Classes that cannot be instantiated on their own and are meant to be subclassed.
- They often contain one or more abstract methods which are methods declared in the abstract class but must be implemented

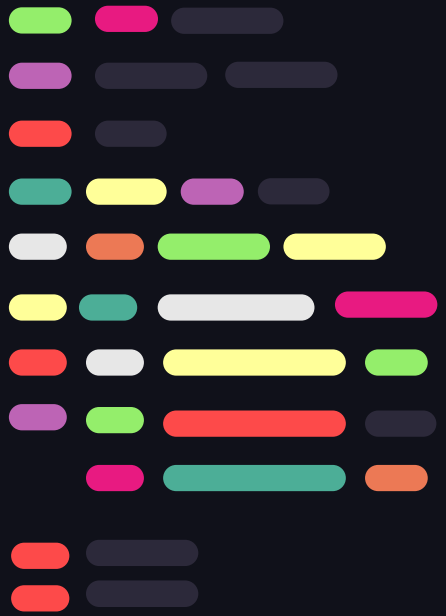
```java
abstract class Animal {
    // abstract Method "does not have a body"
    abstract void Sound();
    //Regular Method
    void eat(){
        System.out.println("Eating...");
    }
}

class Dog extends Animal{
    //Implementing Abstract Method
    void Sound(){
        System.out.println("Woof!");
    }
}

class Cat extends Animal {
    void Sound(){
        System.out.println("Meow !");
    }
}
```

Example!}

```java
class Test{
    public static void main(String[] args) {
        Animal d = new Dog();
        Animal c = new Cat();

        d.Sound();
        d.eat();

        c.Sound();
        c.eat();
    }
}
```

Example!}

Here are eight concepts



1_

- Abstract classes cannot be instantiated directly.

2_

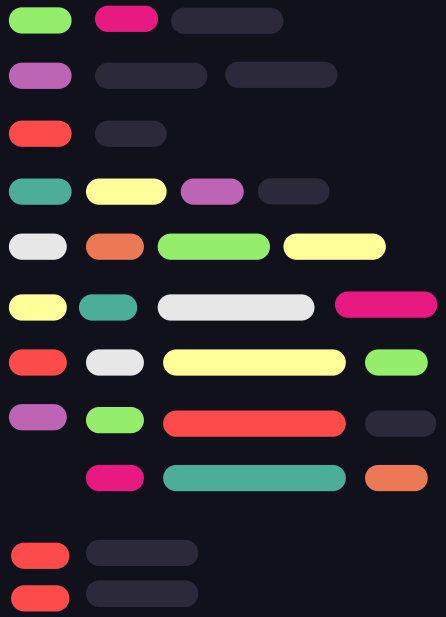
- Abstract classes can contain abstract methods

3_

- Abstract classes can also have concrete methods

4_

- Abstract classes can have constructors



5_

- Abstract classes can have fields and constants

6_

- A subclass that extends an abstract class must implement all abstract methods

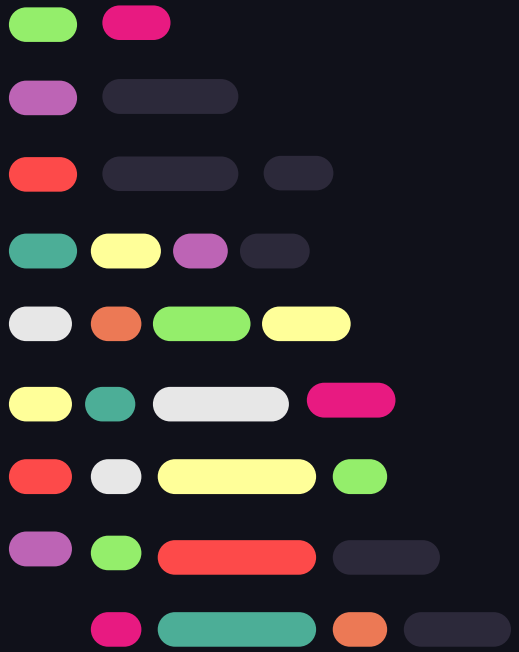
7_

- Abstract classes support polymorphism

8_

- Abstract classes are useful when you want to provide some common functionality

Interface



- An interface in Java is a reference type
- Can contain only constants method signatures, default methods, static methods.
- Interfaces cannot contain instance fields or constructors

```java

interface Animal {
    // abstract methods
    abstract void Sound();
    void eat();
}
//Implementing the interface in the dog class
class Dog extends Animal{
    //Implementing Abstract Method
    void Sound(){
        System.out.println("Woof!");
    }
    void eat(){
        System.out.println("Eating Biscuits"
);
    }
}

interface Sea_Animal extends Animal {
    void Drink();
}

class Dolphin implements Sea_Animal{
    void Drink(){
        System.out.println("Drinking Water");
    }
    void Sound(){
        System.out.println("Click !");
    }
    void eat(){
        System.out.println("Eating Sea Weed"
);
    }
}
```

Example!}

```java
class Test{
    public static void main(String[] args) {
        Animal d = new Dog();
        Animal dolphin = new Dolphin();

        d.Sound();
        d.eat();

        dolphin.Sound();
        dolphin.eat();
        dolphin.Drink();
    }
}
```

Example!}

{ .. Here are six concepts

1_

Interfaces can contain abstract methods

2_

Interfaces can have default methods with a body

3_

A class can implement multiple interfaces

4_

Interfaces cannot have constructors

5_

Interfaces support polymorphism

6_

Ensure that certain methods are implemented

} . .

Interfaces extends Interfaces

Classes implements Interfaces

{..

Awesome Point!

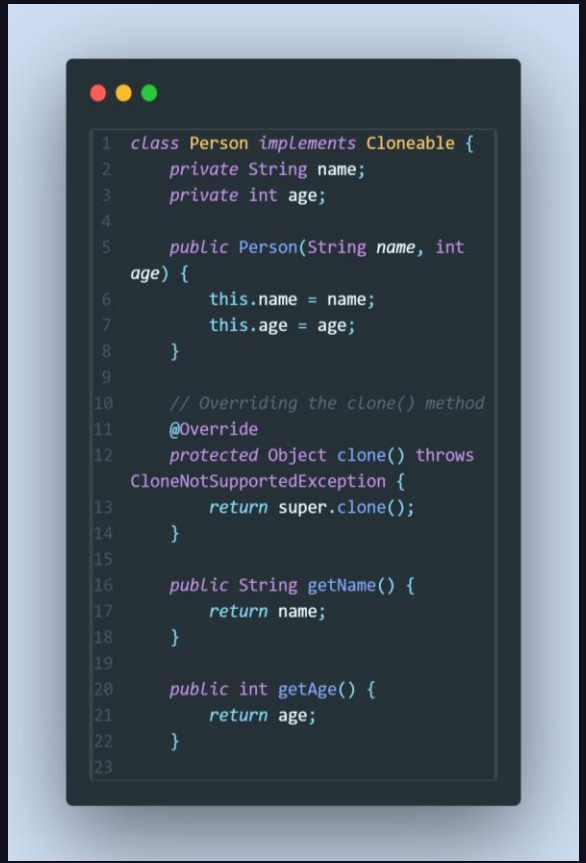
}..

Cloning

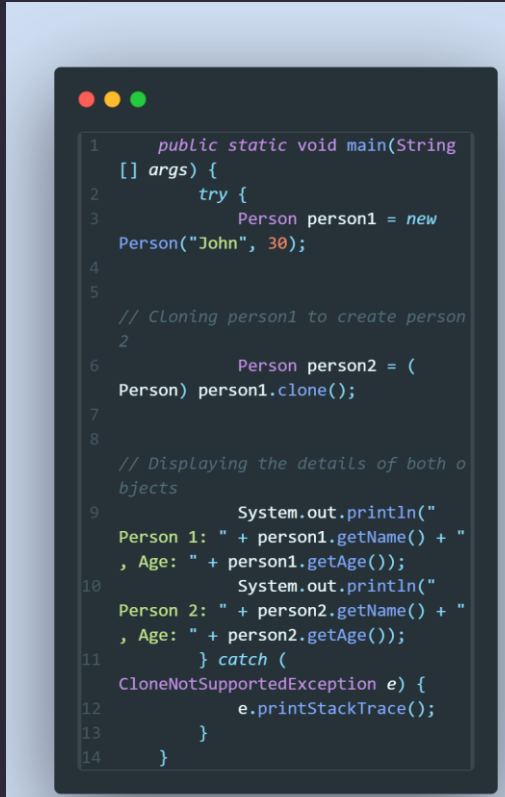
- Cloning in Java is the process of creating an exact copy of an object.
- To enable cloning, a class must implement the Cloneable interface.
- The clone() method from the Object class is used to create a copy of an object.

}..


Lets see an Example!

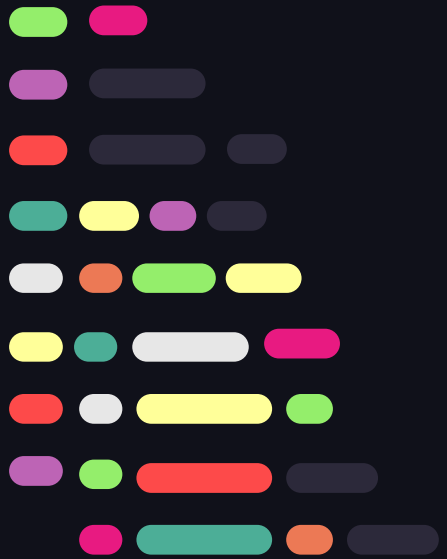


```java
class Person implements Cloneable {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Overriding the clone() method
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

```

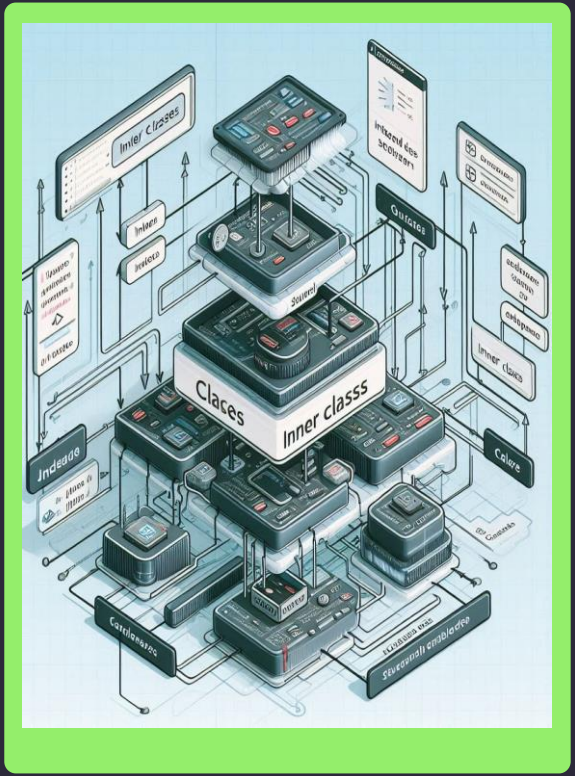


```java
    public static void main(String[] args) {
        try {
            Person person1 = new Person("John", 30);


            // Cloning person1 to create person 2
            Person person2 = (Person) person1.clone();


            // Displaying the details of both objects
            System.out.println("Person 1: " + person1.getName() + ", Age: " + person1.getAge());
            System.out.println("Person 2: " + person2.getName() + ", Age: " + person2.getAge());
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }
```

Inner Classes



- An inner class is a class defined within another class or interface
- There are different types such as:
  - Member inner class
  - Static nested class
  - Local inner class



{ Member inner class

- Member Inner Class: A non-static class defined at the member level of a class

}

Nested Classes

- A static class defined inside another class

Local inner class

- A class defined within a method or a block of code


Lets see an Example!

```java
public class OuterClass {

    // Member Inner Class
    class MemberInnerClass {
        public void display() {
            System.out.println("Inside Member Inner Class.");
        }
    }


// Nested Inner Class (Static Inner Class)
    static class NestedInnerClass {
        public void display() {
            System.out.println("Inside Nested Inner Class.");
        }
    }

    public void createLocalInnerClass() {

// Local Inner Class inside a method
        class LocalInnerClass {
            public void display() {
                System.out.println("Inside Local Inner Class.");
            }
        }
```

```java

        LocalInnerClass localInner = new LocalInnerClass();
        localInner.display();
    }

    public void showInnerClassMessages() {
        // Using the Member Inner Class
        MemberInnerClass memberInner = new MemberInnerClass();
        memberInner.display();

        // Using the Nested Inner Class
        NestedInnerClass nestedInner = new NestedInnerClass();
        nestedInner.display();

        // Using the Local Inner Class
        createLocalInnerClass();
    }

    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.showInnerClassMessages();
    }
}
```

Resources

https://www.w3schools.com/java/java_ref_string.asp
https://www.geeksforgeeks.org/inner-class-java/
https://www.javatpoint.com/object-cloning
https://www.javatpoint.com/abstract-class-in-java
https://www.w3schools.com/java/java_interface.asp

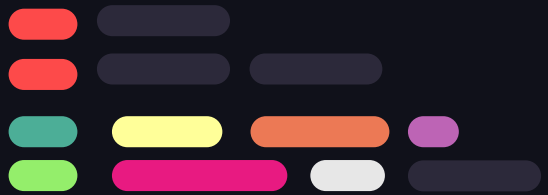
Alternative resources

Dr. Mojtaba Vahidi Asl SLides

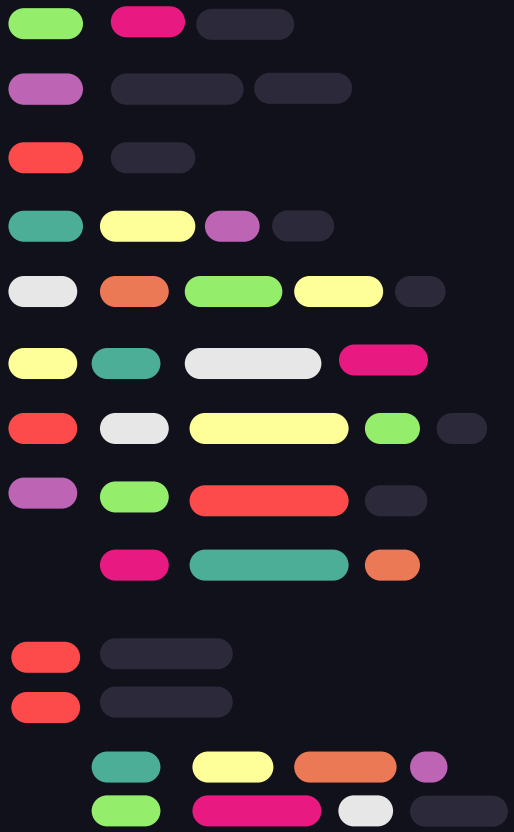
{

Any Questions?

< I will be happy to answer :) >



}



Thanks!