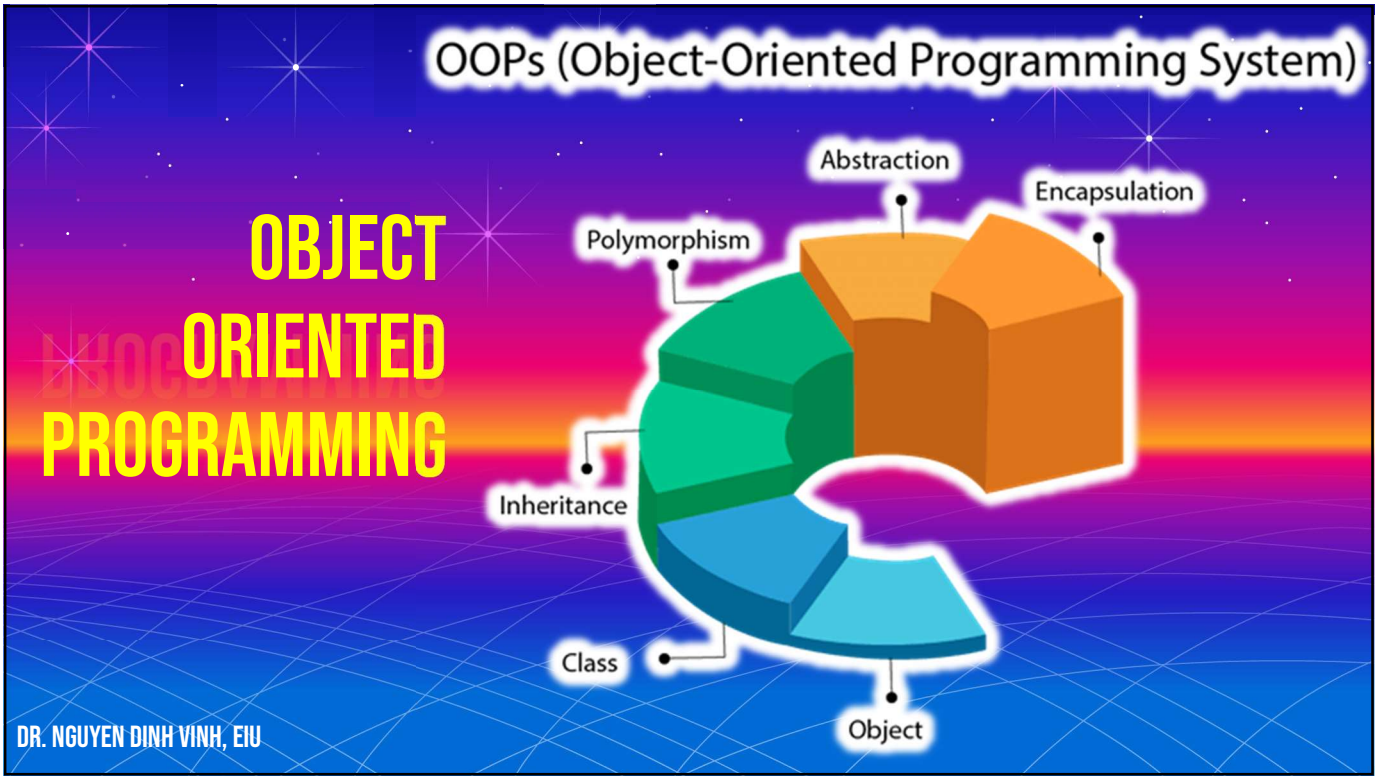

EIU
TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY
CSE203 – OBJECT ORIENTED PROGRAMMING
ENCAPSULATION – ABSTRACTION – INHERITANCE - POLYMORPHISM
Mobile programming

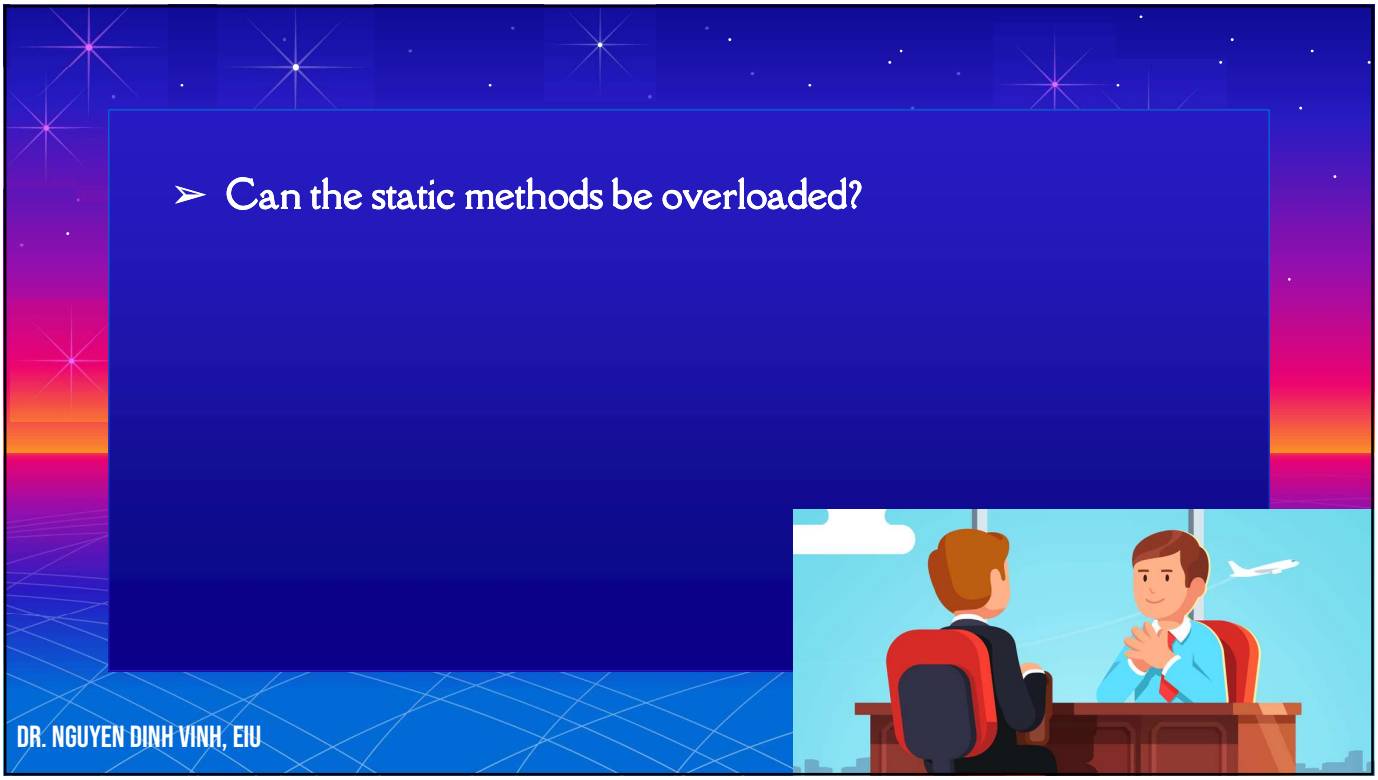
1


OOPs (Object-Oriented Programming System)
OBJECT ORIENTED PROGRAMMING
Abstraction
Encapsulation
Polymorphism
Inheritance
Class
Object
DR. NGUYEN DINH VINH, EIU

2


"WHY DO YOU WANT THIS JOB?"
"WHAT'S YOUR BIGGEST WEAKNESS?"
COMMON INTERVIEW QUESTIONS
(And How to Answer Them!)
"WHY SHOULD WE HIRE YOU?"
"WHAT ARE YOUR SALARY EXPECTATIONS?"
DR. NGUYEN DINH VINH, EIU

3


➢ Can the static methods be overloaded?
DR. NGUYEN DINH VINH, EIU

7

- Difference between static methods, static variables, and static classes in java?

DR. NGUYEN DINH VINH, EIU

8

- What are the differences between constructor and method of a class in Java?

DR. NGUYEN DINH VINH, EIU

11

- Can you call a constructor of a class inside the another constructor?

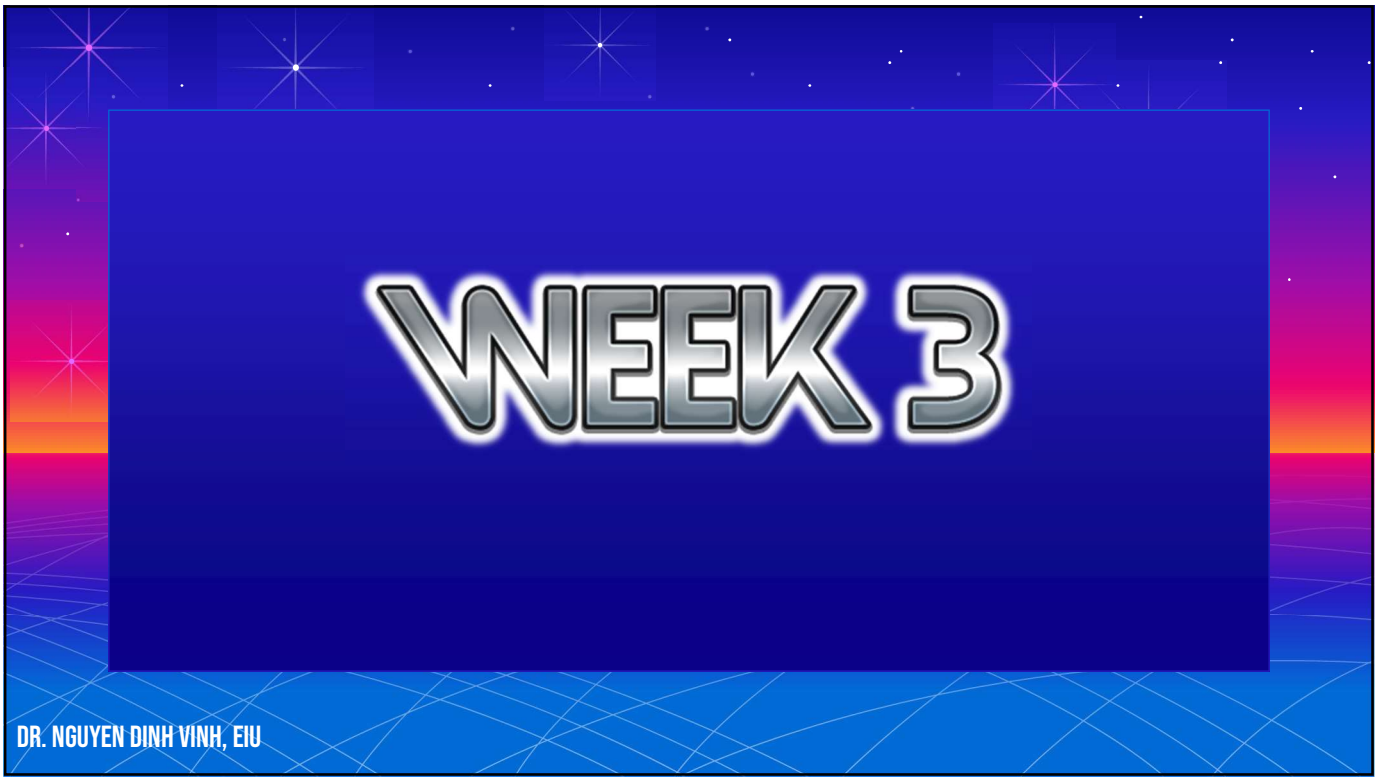
13

Constructor Chaining

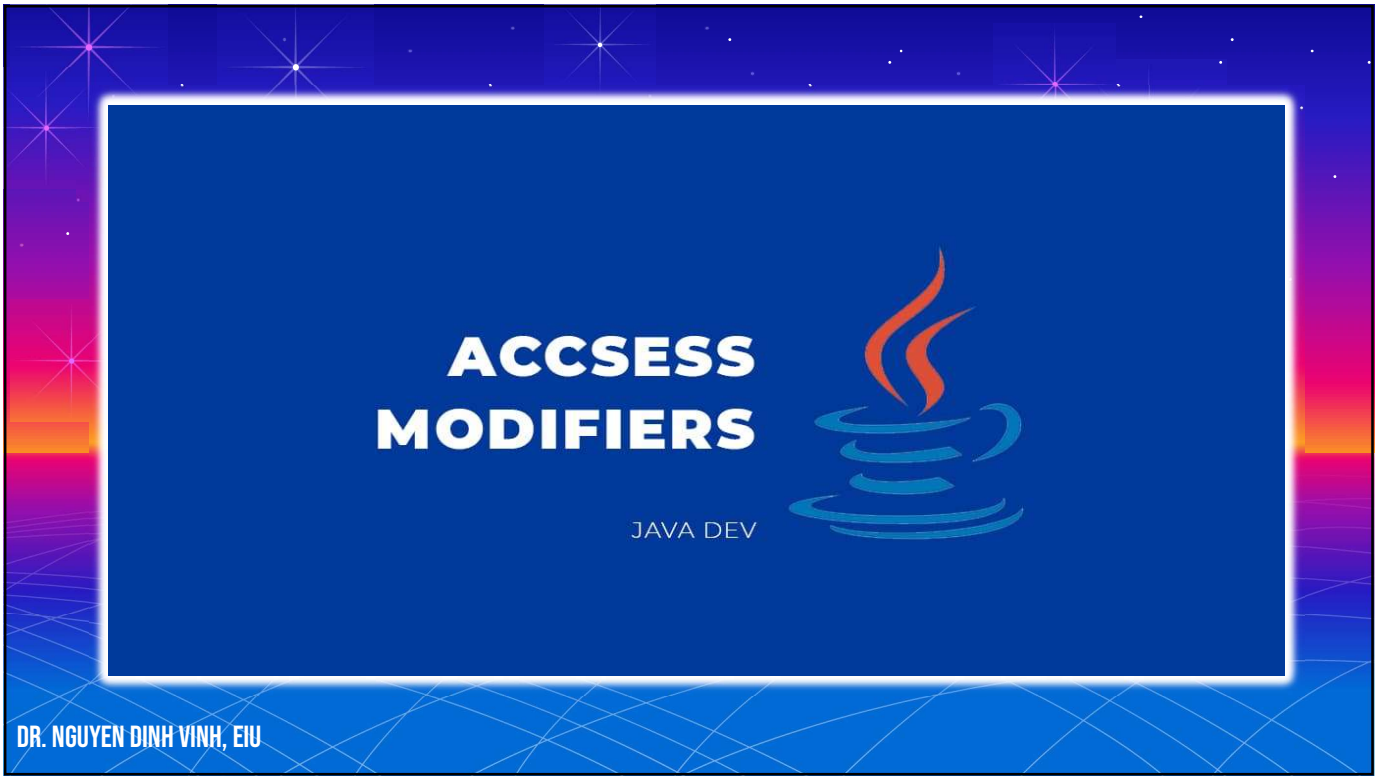
```
public class InterviewBit{

    InterviewBit(){
        this("Hi InterviewBit!");
    }

    InterviewBit(String s){
        System.out.println(s);
    }

    public static void main(String[]args){
        InterviewBit ib = new InterviewBit();
    }
}
```


InterviewBit

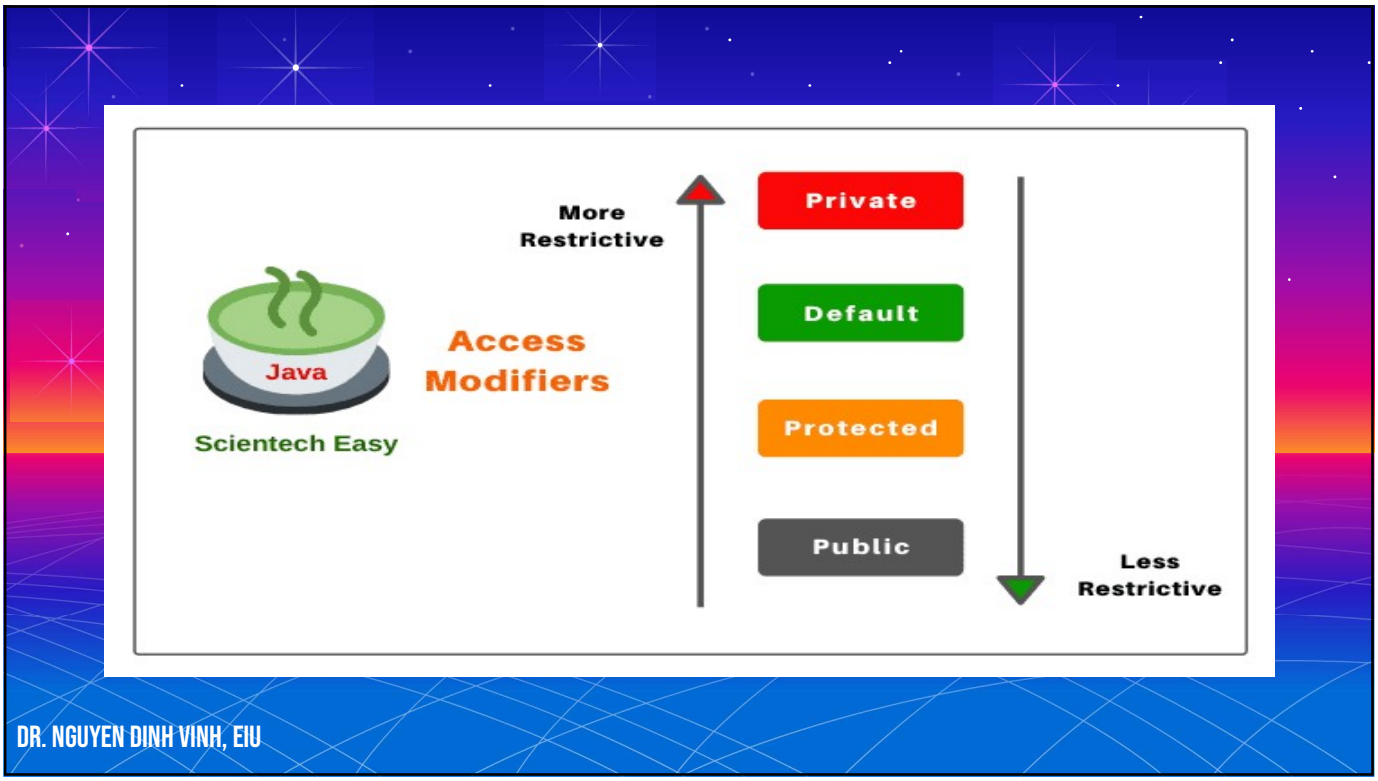
14


WEEK 3
DR. NGUYEN DINH VINH, EIU

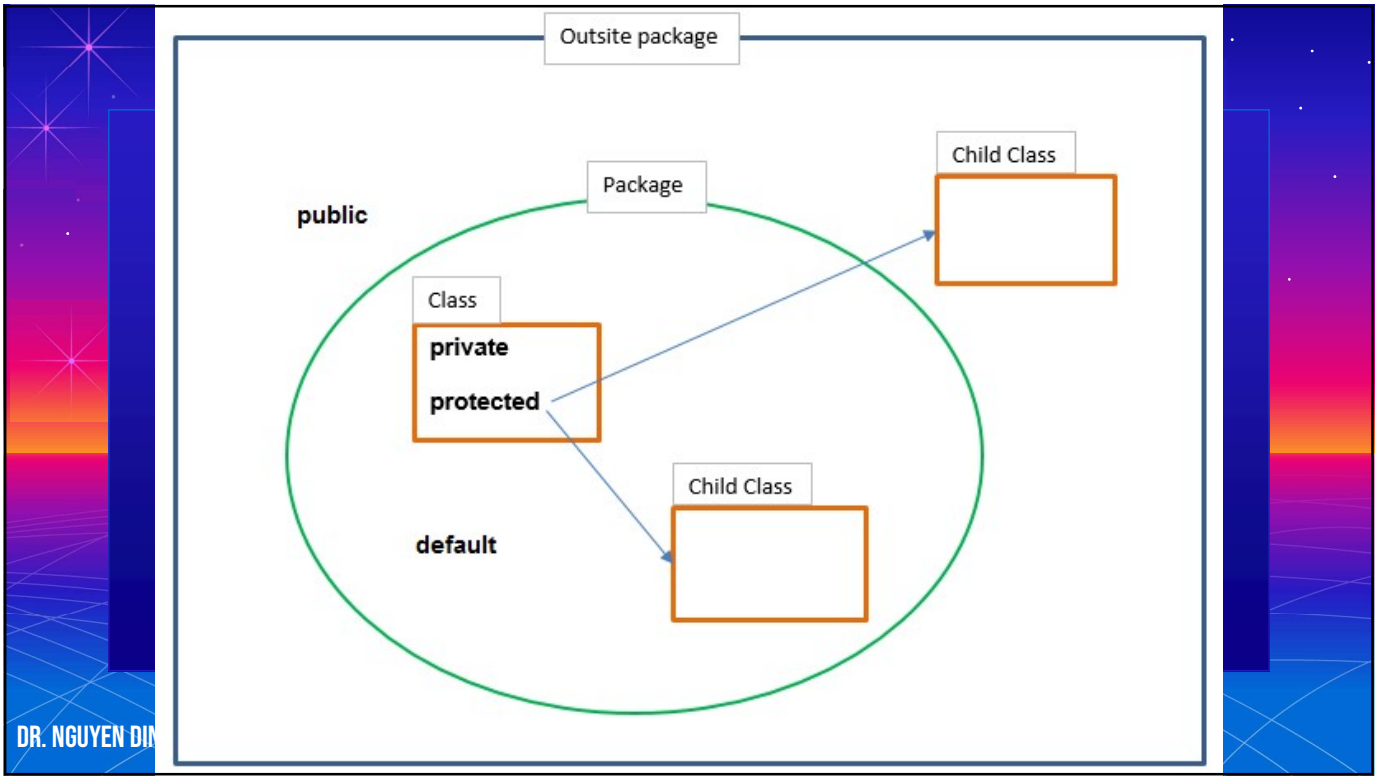
15


ACCSESS
MODIFIERS
JAVA DEV
DR. NGUYEN DINH VINH, EIU

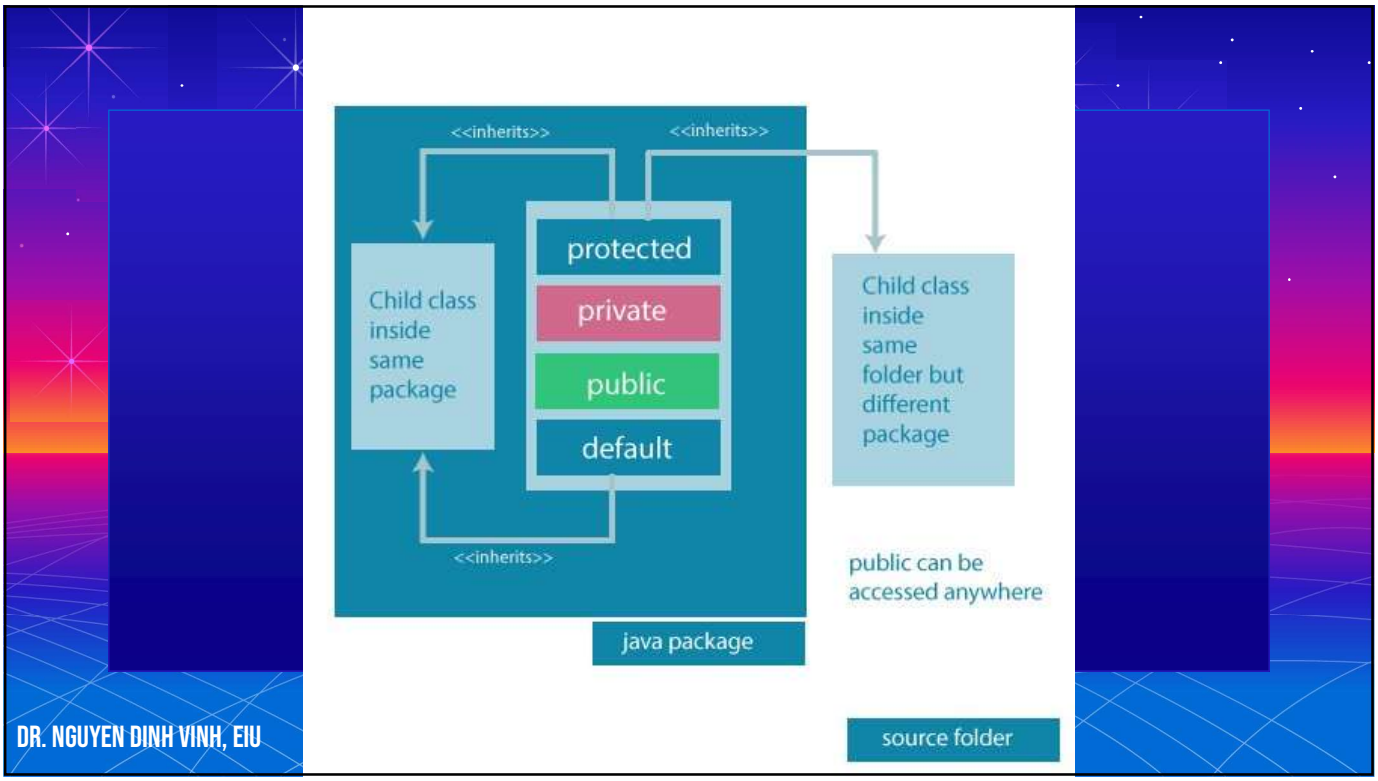
27


More
Restrictive
Private
Default
Protected
Public
Less
Restrictive
Java
Scientech Easy
Access
Modifiers
DR. NGUYEN DINH VINH, EIU

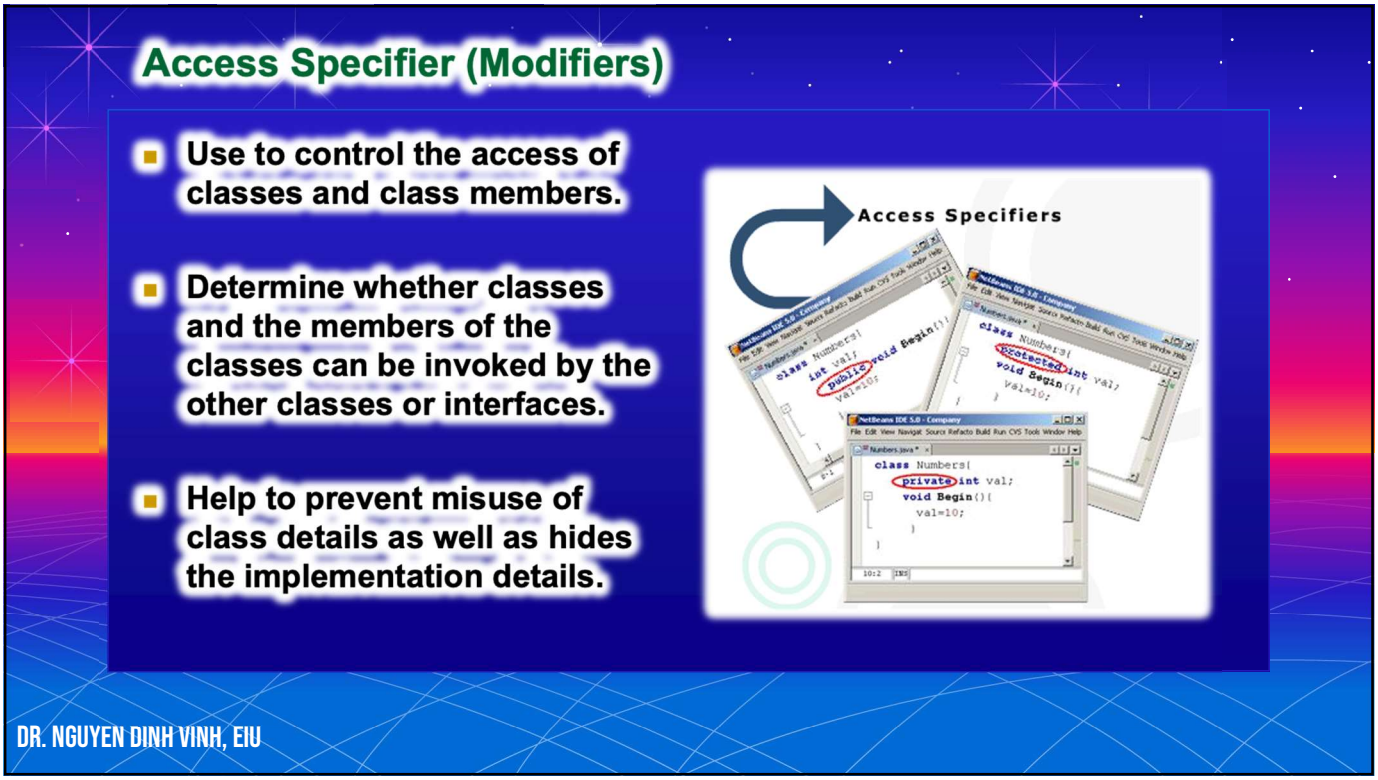
28


Outsite package
Package
public
Class
private
protected
Child Class
Child Class
default
DR. NGUYEN DI

29


<<inherits>>
<<inherits>>
protected
private
public
default
Child class inside same package
Child class inside same folder but different package
<<inherits>>
public can be accessed anywhere
java package
source folder
DR. NGUYEN DINH VINH, EIU

30


Access Specifier (Modifiers)
Use to control the access of classes and class members.
Determine whether classes and the members of the classes can be invoked by the other classes or interfaces.
Help to prevent misuse of class details as well as hides the implementation details.
Access Specifiers
DR. NGUYEN DINH VINH, EIU

31

Java Access modifiers (class)

32

At the top level or class level – public, or *package-private* (no explicit modifier) is allowed.

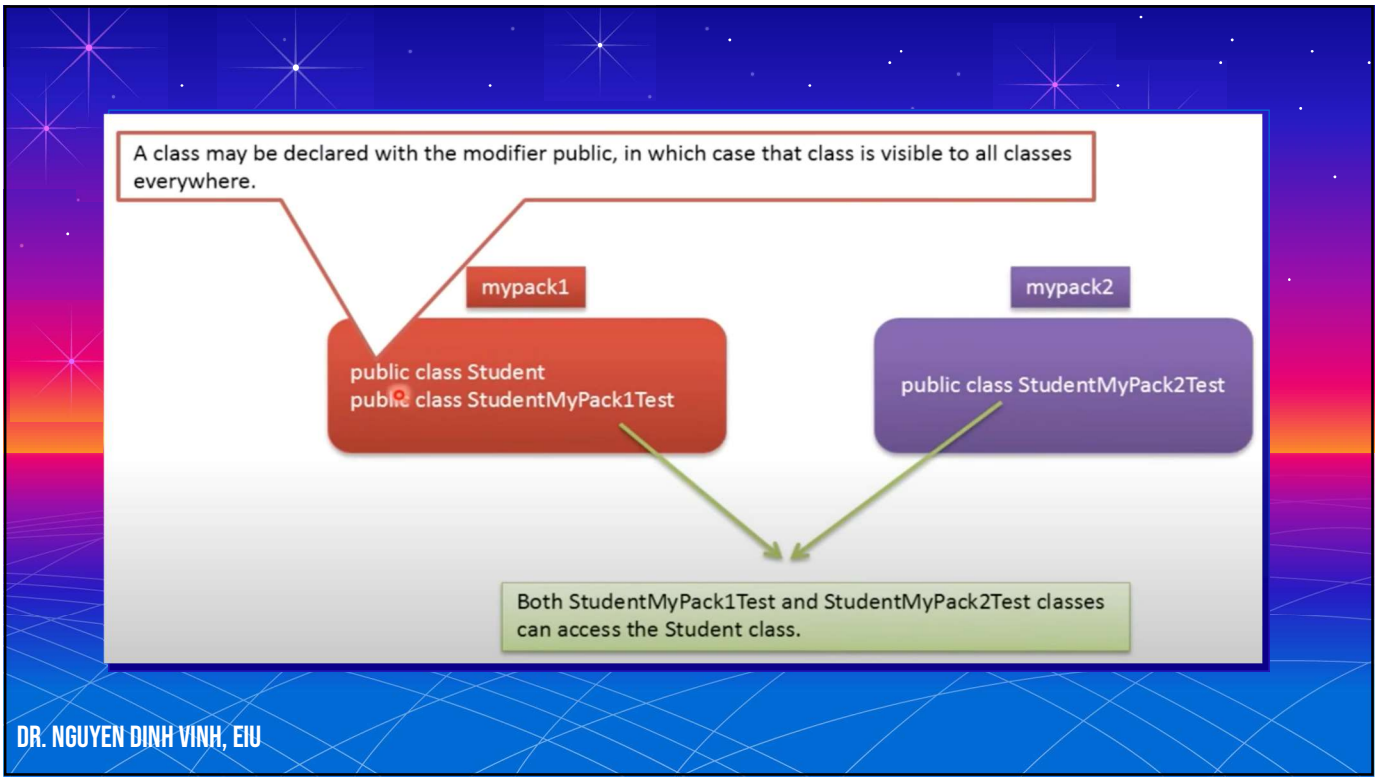
```
public class Student
{

}
```

```
class Student
{

}
```

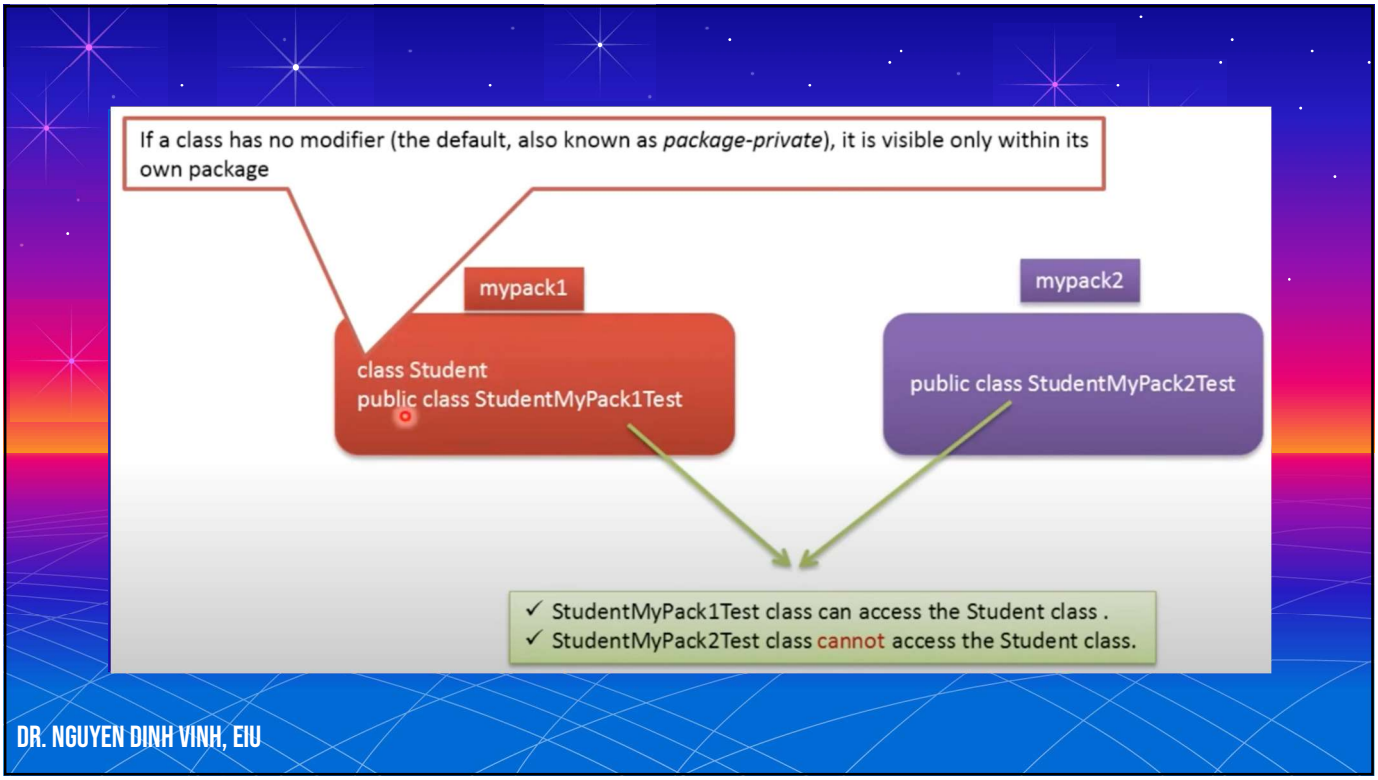
```
protected class Student
{

}
```

```
private class Student
{

}
```

33


A class may be declared with the modifier public, in which case that class is visible to all classes everywhere.
mypack1
public class Student
public class StudentMyPack1Test
mypack2
public class StudentMyPack2Test
Both StudentMyPack1Test and StudentMyPack2Test classes can access the Student class.
DR. NGUYEN DINH VINH, EIU

34


If a class has no modifier (the default, also known as *package-private*), it is visible only within its own package
mypack1
class Student
public class StudentMyPack1Test
mypack2
public class StudentMyPack2Test
✓ StudentMyPack1Test class can access the Student class .
✓ StudentMyPack2Test class cannot access the Student class.
DR. NGUYEN DINH VINH, EIU

35

Tips on Choosing an Access Level

- **Use the most restrictive access level that makes sense for a particular member.**
  - Use private unless you have a good reason not to.
- **Avoid public fields except for constants.**
  - Public fields tend to link you to a particular implementation and limit your flexibility in changing your code.

DR. NGUYEN DINH VINH, EIU

36

Java Access modifiers (field)

At the **field/attribute** level —public, private, protected, or package-private (no explicit modifier) is allowed.

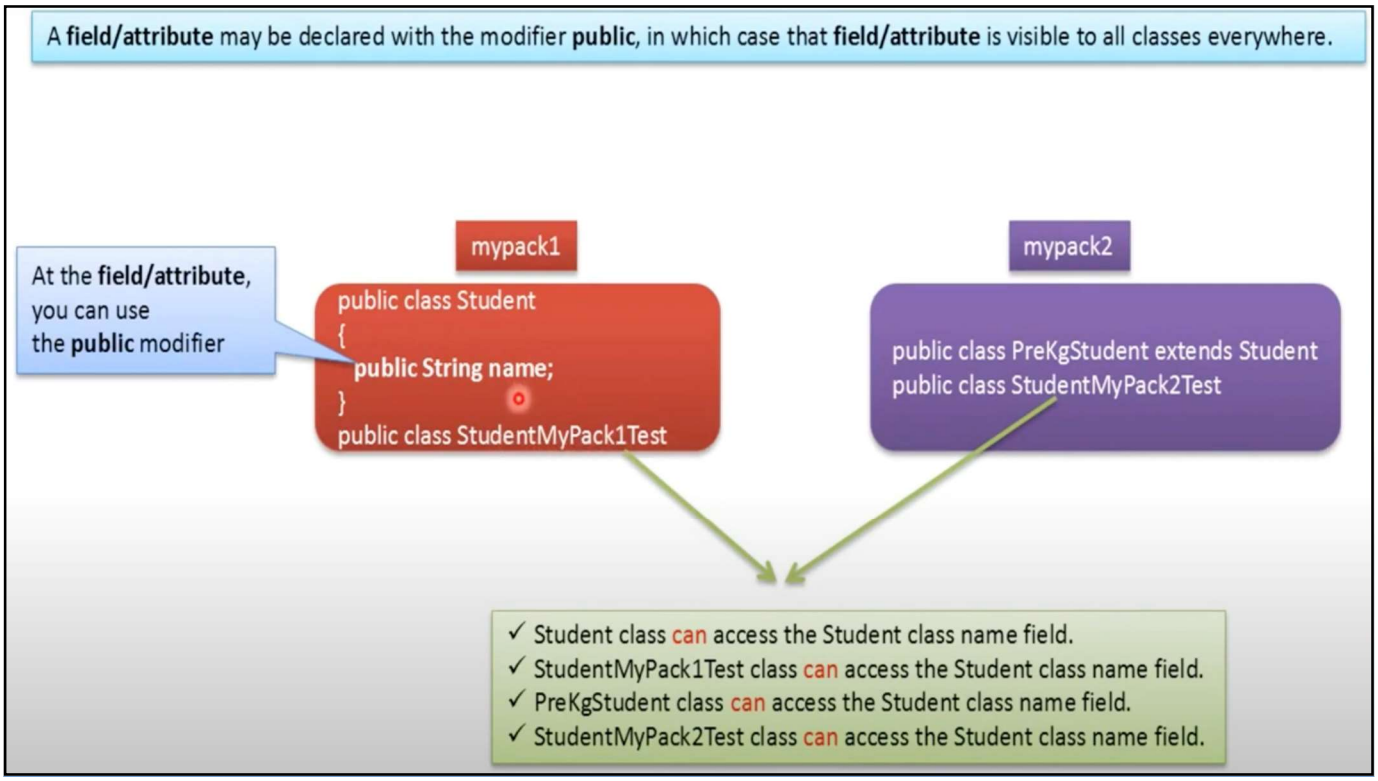
```
public class Student
{
   public String name;
}
```

```
public class Student
{
   private String name;
}
```

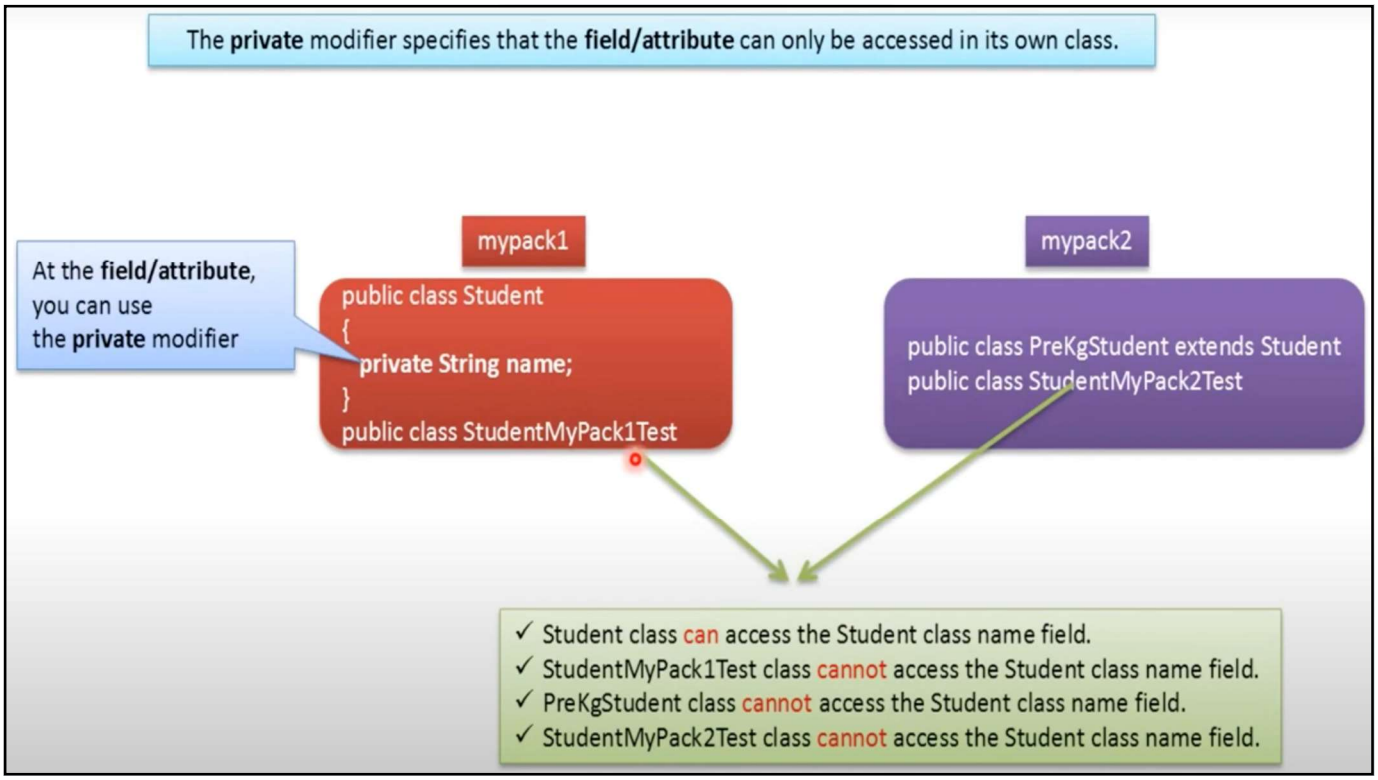
```
public class Student
{
   protected String name;
}
```

```
public class Student
{
   String name;
}
```

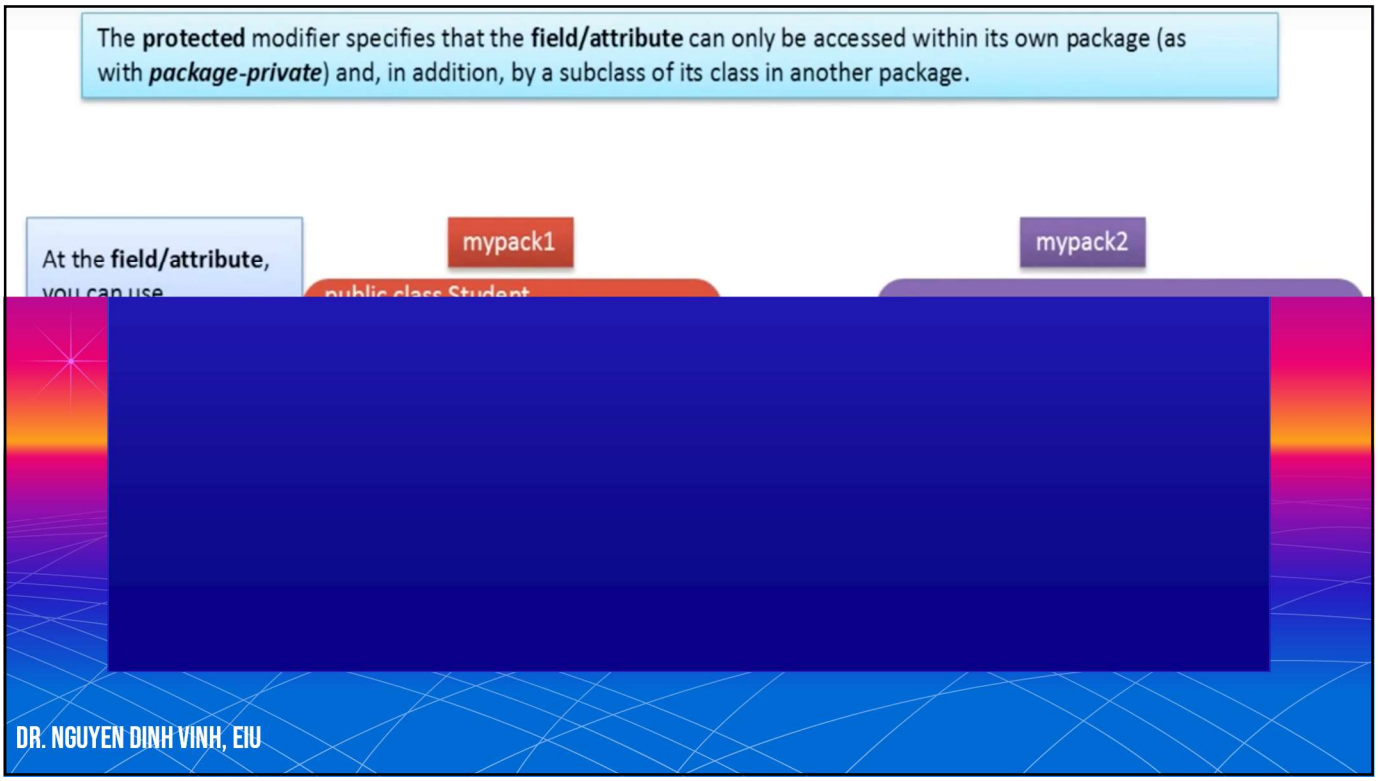
37


A field/attribute may be declared with the modifier public, in which case that field/attribute is visible to all classes everywhere.
At the field/attribute, you can use the public modifier
mypack1
public class Student
{
public String name;
}
public class StudentMyPack1Test
mypack2
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class name field.
✓ StudentMyPack1Test class can access the Student class name field.
✓ PreKgStudent class can access the Student class name field.
✓ StudentMyPack2Test class can access the Student class name field.

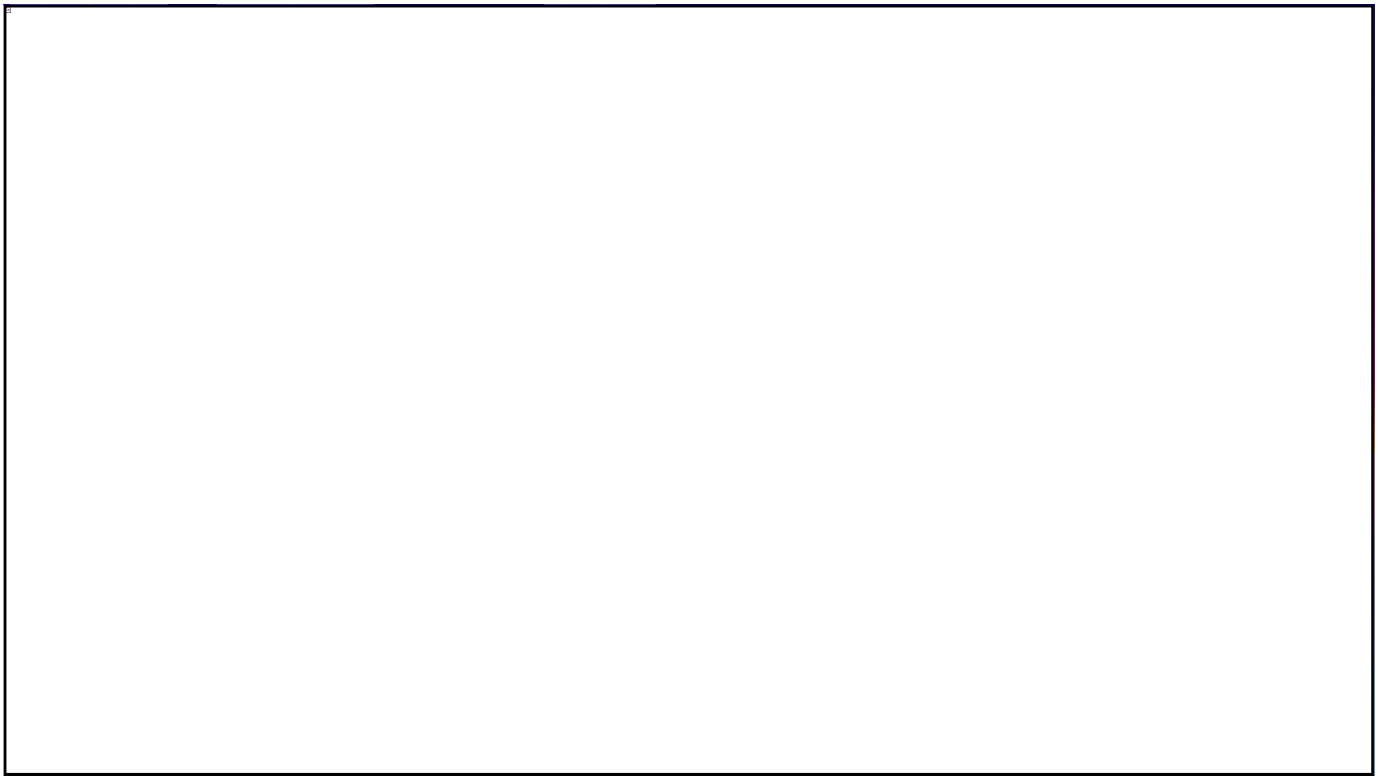
38


The private modifier specifies that the field/attribute can only be accessed in its own class.
At the field/attribute, you can use the private modifier
mypack1
public class Student
{
private String name;
}
public class StudentMyPack1Test
mypack2
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class name field.
✓ StudentMyPack1Test class cannot access the Student class name field.
✓ PreKgStudent class cannot access the Student class name field.
✓ StudentMyPack2Test class cannot access the Student class name field.

39


The protected modifier specifies that the field/attribute can only be accessed within its own package (as with package-private) and, in addition, by a subclass of its class in another package.
At the field/attribute,
mypack1
mypack2
DR. NGUYEN DINH VINH, EIU

40



41

Java Access modifiers (method)

42

At the method level —public, private, protected, or package-private (no explicit modifier) is allowed.

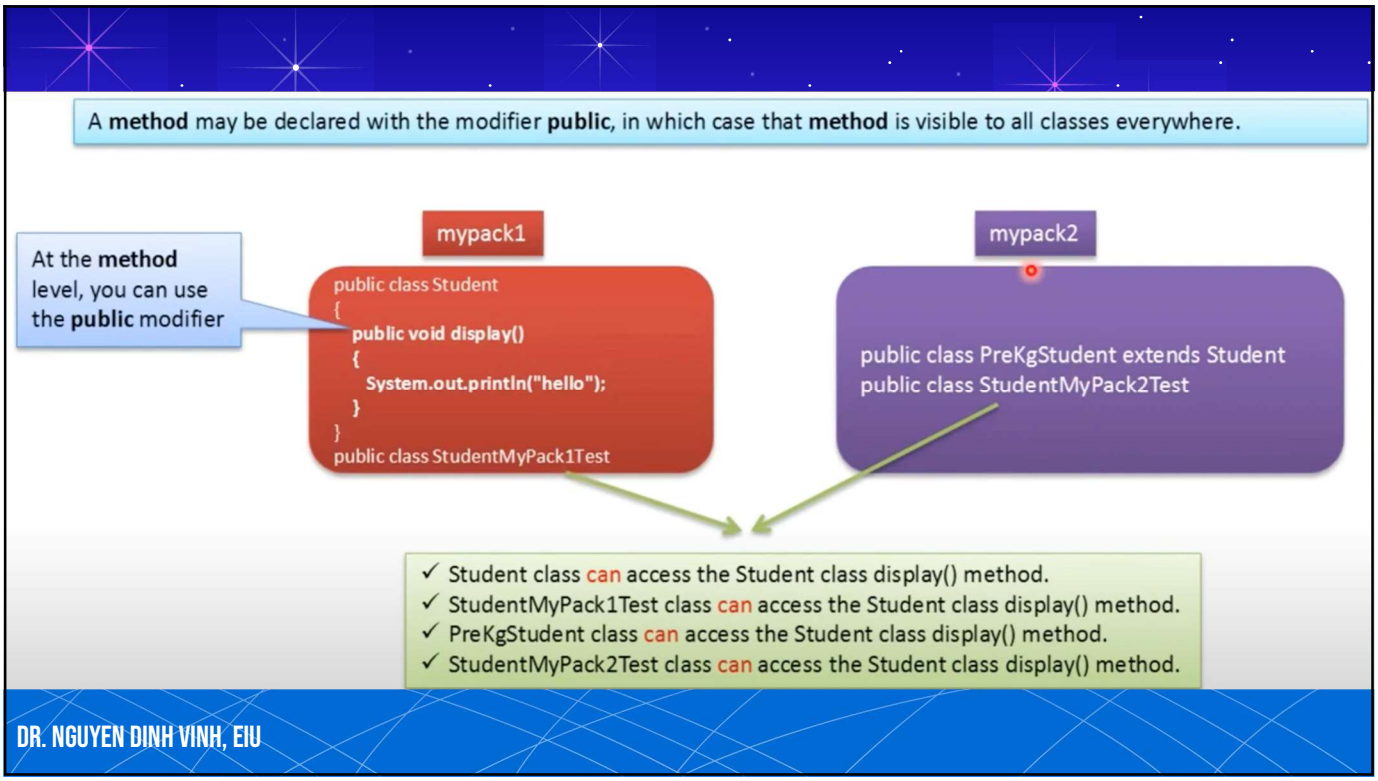
```
public class Student
{
   public void display()
   {
     System.out.println("hello");
   }
}
```

```
public class Student
{
   private void display()
   {
     System.out.println("hello");
   }
}
```

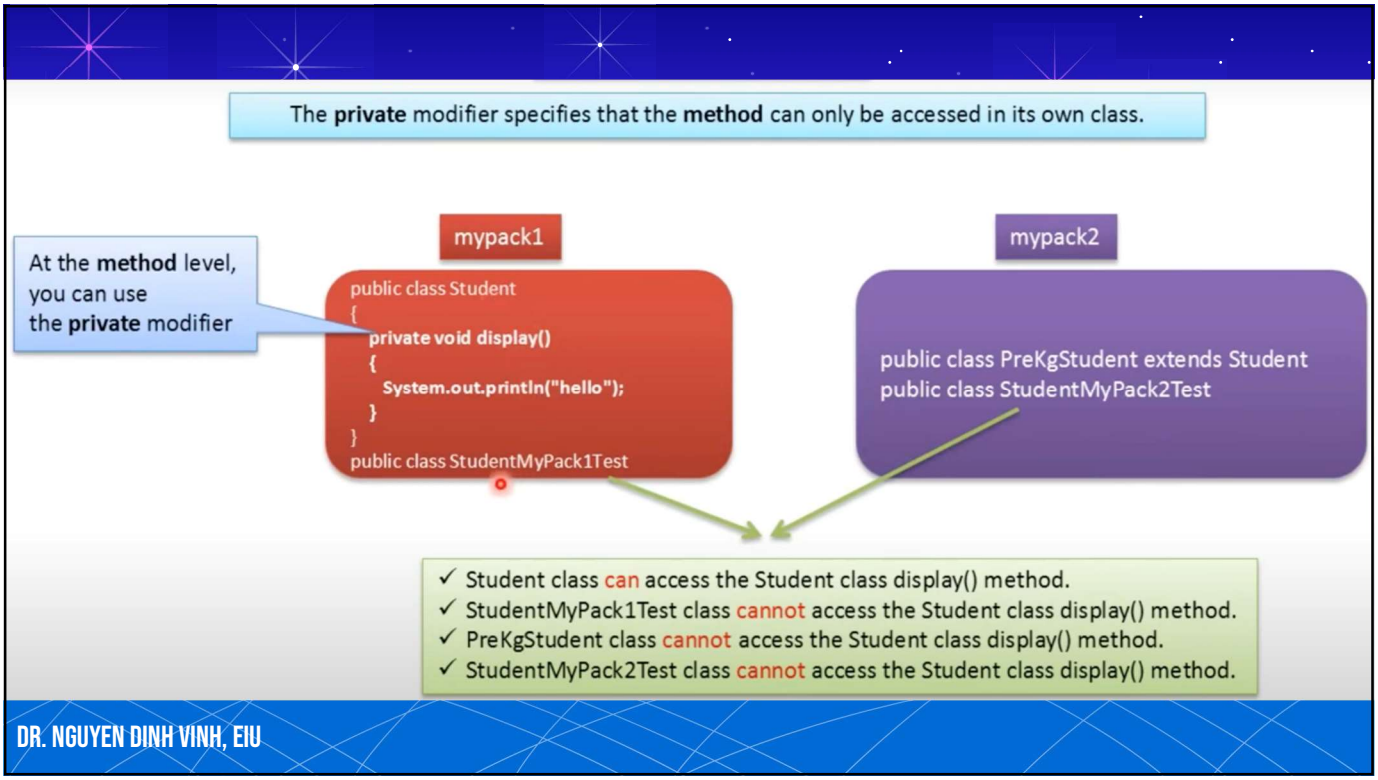
```
public class Student
{
   protected void display()
   {
     System.out.println("hello");
   }
}
```

```
public class Student
{
    void display()
   {
     System.out.println("hello");
   }
}
```

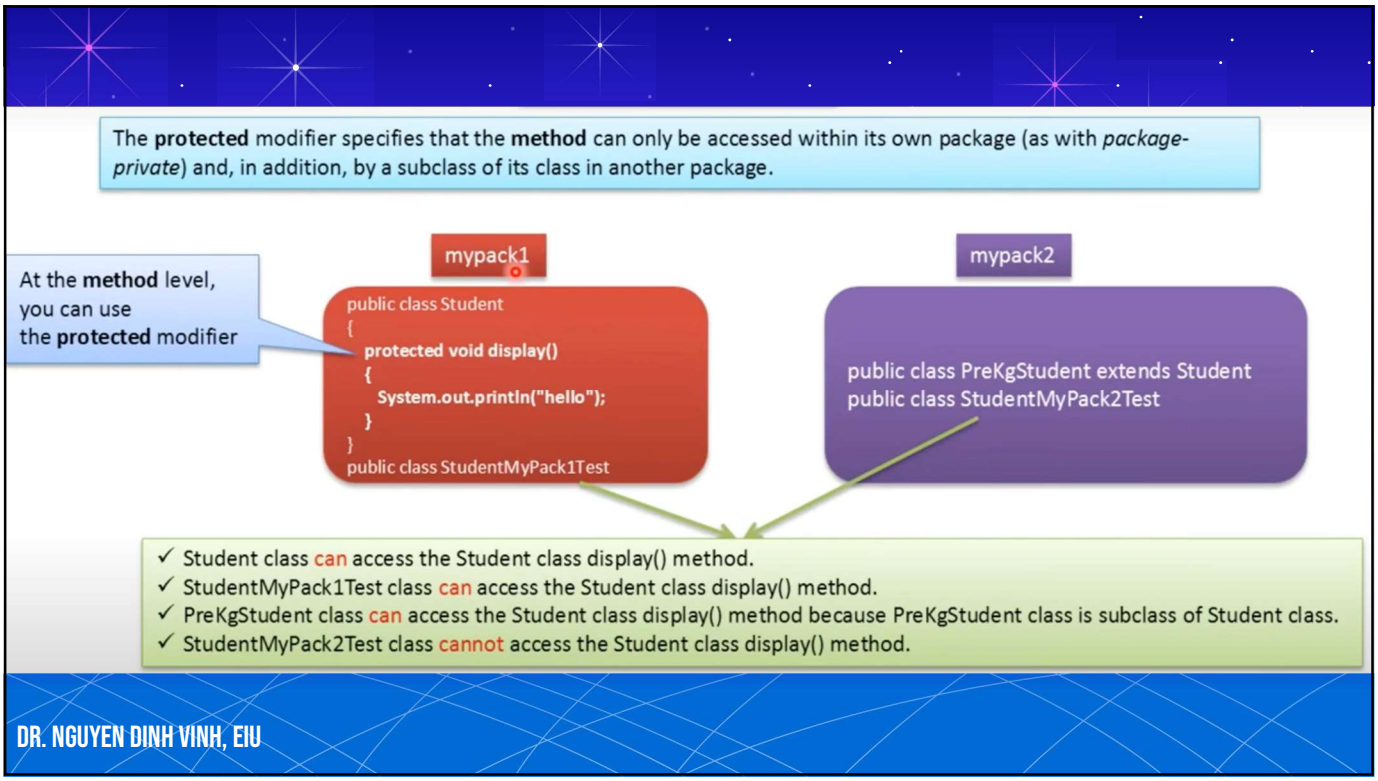

43


A method may be declared with the modifier public, in which case that method is visible to all classes everywhere.
At the method level, you can use the public modifier
mypack1
public class Student
{
public void display()
{
System.out.println("hello");
}
}
public class StudentMyPack1Test
mypack2
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class display() method.
✓ StudentMyPack1Test class can access the Student class display() method.
✓ PreKgStudent class can access the Student class display() method.
✓ StudentMyPack2Test class can access the Student class display() method.
DR. NGUYEN DINH VINH, EIU

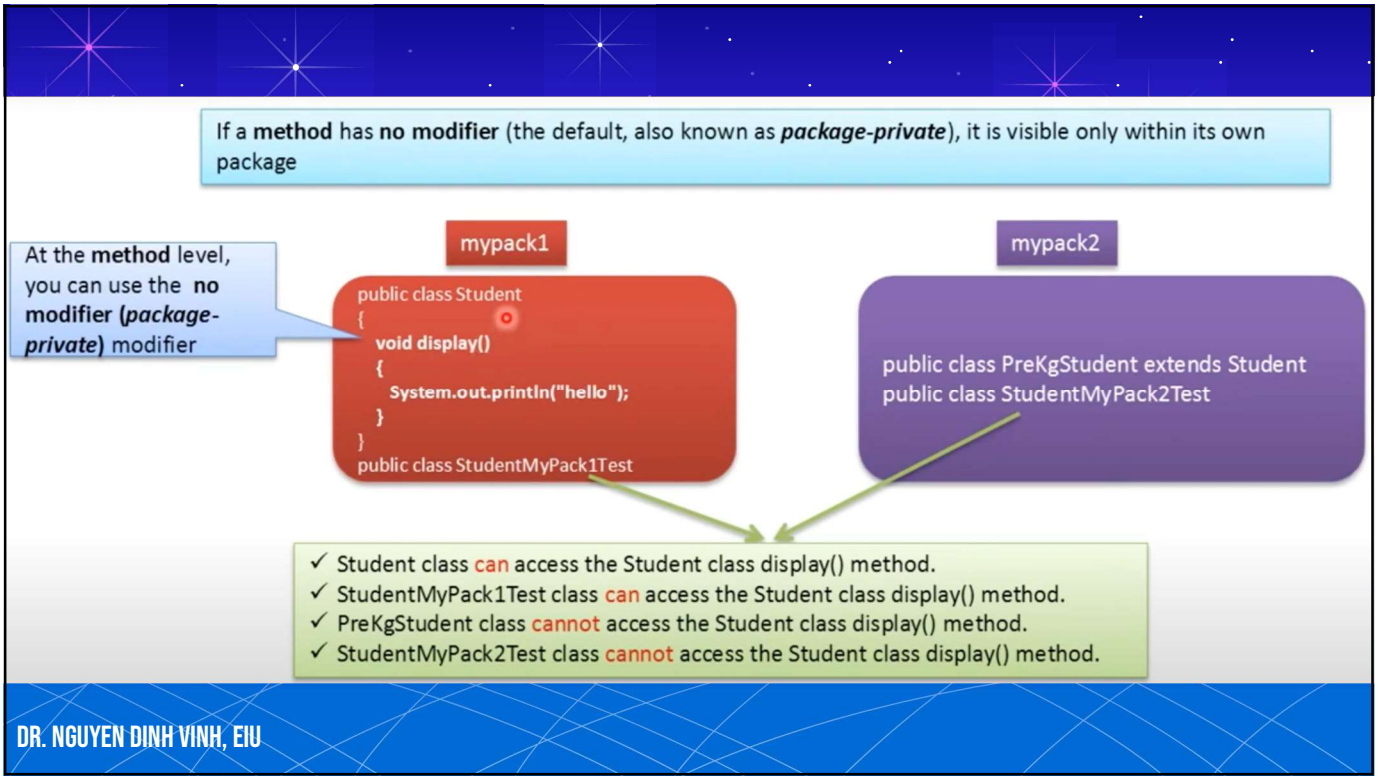
44


The private modifier specifies that the method can only be accessed in its own class.
At the method level, you can use the private modifier
mypack1
public class Student
{
private void display()
{
System.out.println("hello");
}
}
public class StudentMyPack1Test
mypack2
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class display() method.
✓ StudentMyPack1Test class cannot access the Student class display() method.
✓ PreKgStudent class cannot access the Student class display() method.
✓ StudentMyPack2Test class cannot access the Student class display() method.
DR. NGUYEN DINH VINH, EIU

45


The protected modifier specifies that the method can only be accessed within its own package (as with package-private) and, in addition, by a subclass of its class in another package.
At the method level, you can use the protected modifier
mypack1
public class Student
{
protected void display()
{
System.out.println("hello");
}
}
public class StudentMyPack1Test
mypack2
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class display() method.
✓ StudentMyPack1Test class can access the Student class display() method.
✓ PreKgStudent class can access the Student class display() method because PreKgStudent class is subclass of Student class.
✓ StudentMyPack2Test class cannot access the Student class display() method.
DR. NGUYEN DINH VINH, EIU

46


If a method has no modifier (the default, also known as package-private), it is visible only within its own package
At the method level, you can use the no modifier (package-private) modifier
mypack1
public class Student
{
void display()
{
System.out.println("hello");
}
}
public class StudentMyPack1Test
mypack2
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class display() method.
✓ StudentMyPack1Test class can access the Student class display() method.
✓ PreKgStudent class cannot access the Student class display() method.
✓ StudentMyPack2Test class cannot access the Student class display() method.
DR. NGUYEN DINH VINH, EIU

47

Java Access modifiers (constructor)

48

At the Constructor level —public, private, protected, or package-private (no explicit modifier) is allowed.

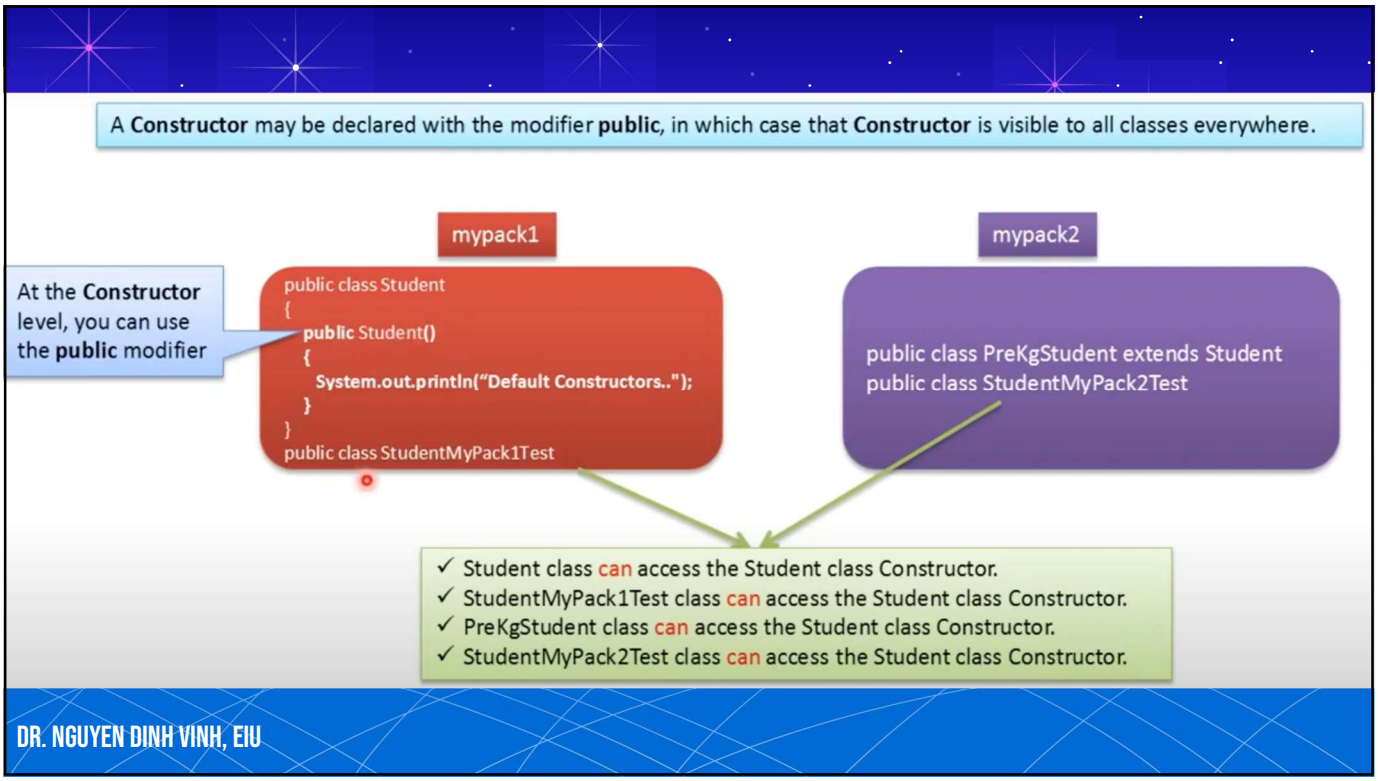
```
public class Student
{
    public Student()
    {
        System.out.println("Default constructor ...");
    }
}
```

```
public class Student
{
    private Student()
    {
        System.out.println("Default constructor ...");
    }
}
```

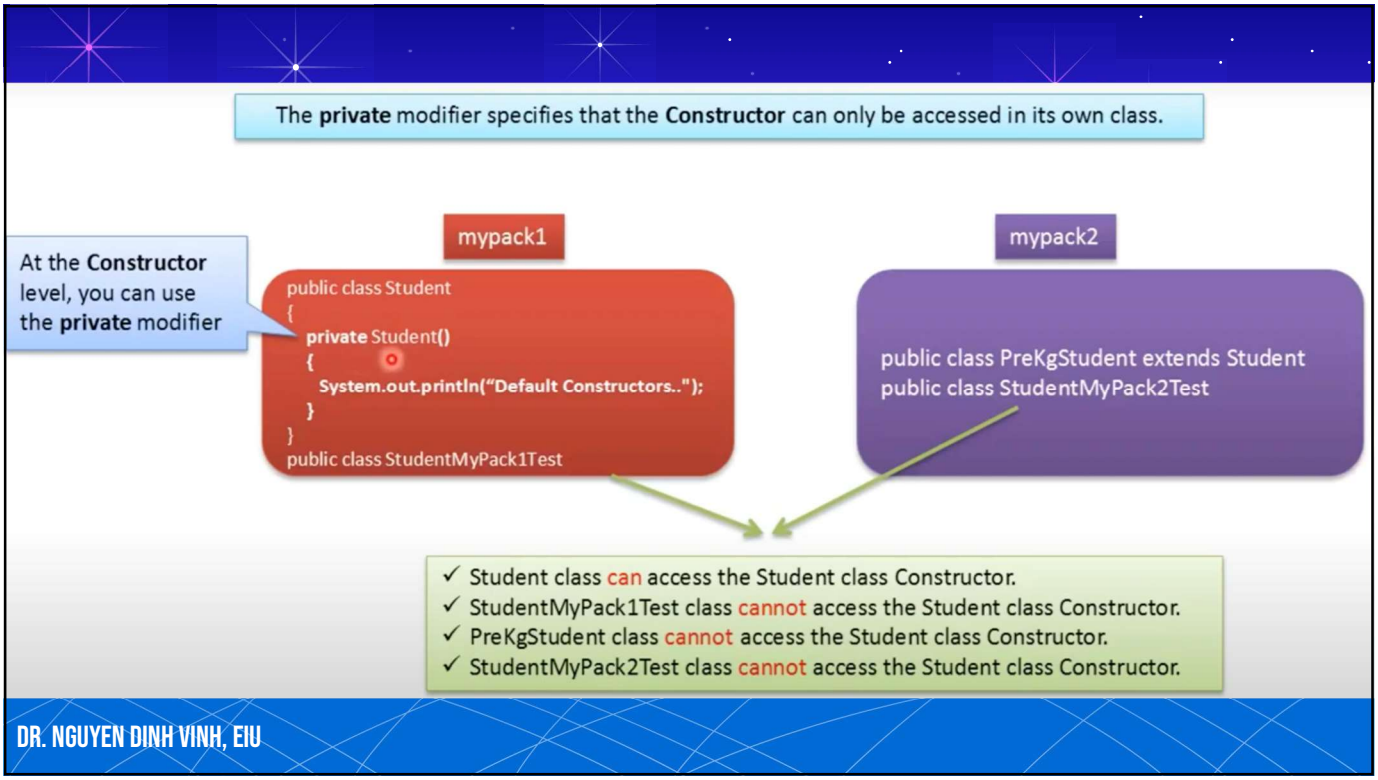
```
public class Student
{
    protected Student()
    {
        System.out.println("Default constructor ...");
    }
}
```

```
public class Student
{
    Student()
    {
        System.out.println("Default constructor ...");
    }
}
```

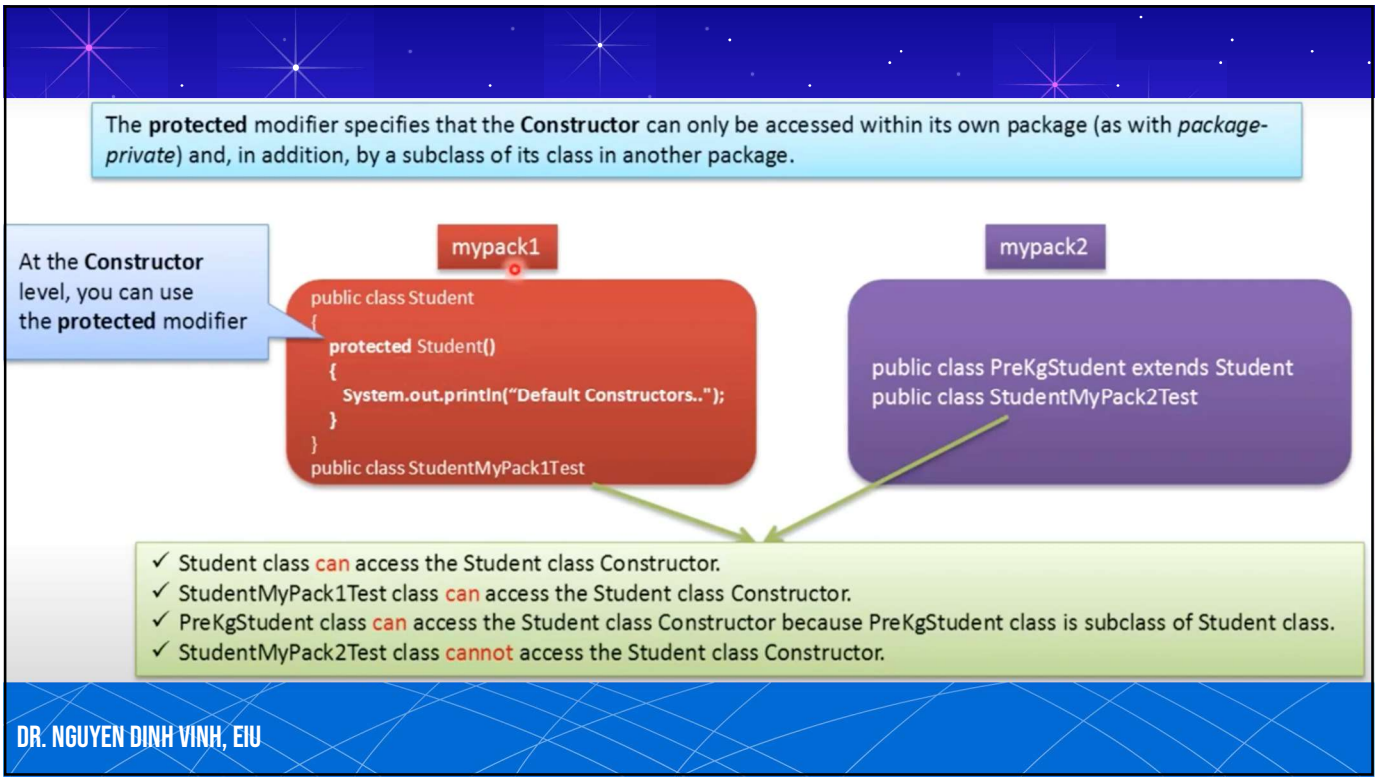
49


A Constructor may be declared with the modifier public, in which case that Constructor is visible to all classes everywhere.
mypack1
mypack2
At the Constructor level, you can use the public modifier
public class Student
{
public Student()
{
System.out.println("Default Constructors..");
}
}
public class StudentMyPack1Test
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class Constructor.
✓ StudentMyPack1Test class can access the Student class Constructor.
✓ PreKgStudent class can access the Student class Constructor.
✓ StudentMyPack2Test class can access the Student class Constructor.
DR. NGUYEN DINH VINH, EIU

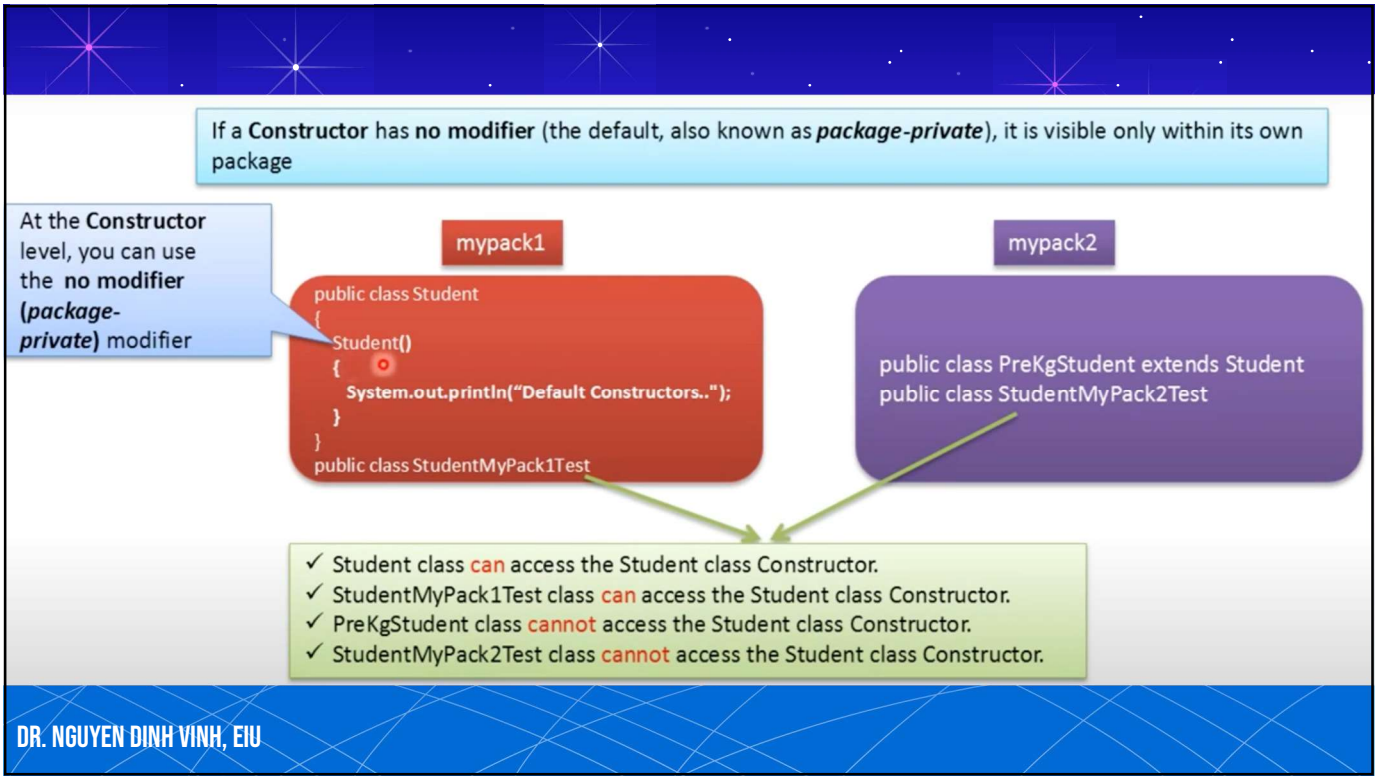
50


The private modifier specifies that the Constructor can only be accessed in its own class.
mypack1
mypack2
At the Constructor level, you can use the private modifier
public class Student
{
private Student()
{
System.out.println("Default Constructors..");
}
}
public class StudentMyPack1Test
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class Constructor.
✓ StudentMyPack1Test class cannot access the Student class Constructor.
✓ PreKgStudent class cannot access the Student class Constructor.
✓ StudentMyPack2Test class cannot access the Student class Constructor.
DR. NGUYEN DINH VINH, EIU

51


The protected modifier specifies that the Constructor can only be accessed within its own package (as with package-private) and, in addition, by a subclass of its class in another package.
At the Constructor level, you can use the protected modifier
mypack1
public class Student
{
protected Student()
{
System.out.println("Default Constructors..");
}
}
public class StudentMyPack1Test
mypack2
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class Constructor.
✓ StudentMyPack1Test class can access the Student class Constructor.
✓ PreKgStudent class can access the Student class Constructor because PreKgStudent class is subclass of Student class.
✓ StudentMyPack2Test class cannot access the Student class Constructor.
DR. NGUYEN DINH VINH, EIU

52


If a Constructor has no modifier (the default, also known as package-private), it is visible only within its own package
At the Constructor level, you can use the no modifier (package-private) modifier
mypack1
public class Student
{
Student()
{
System.out.println("Default Constructors..");
}
}
public class StudentMyPack1Test
mypack2
public class PreKgStudent extends Student
public class StudentMyPack2Test
✓ Student class can access the Student class Constructor.
✓ StudentMyPack1Test class can access the Student class Constructor.
✓ PreKgStudent class cannot access the Student class Constructor.
✓ StudentMyPack2Test class cannot access the Student class Constructor.
DR. NGUYEN DINH VINH, EIU

53

What is Access Levels in Java? | Java Access Levels

DR. NGUYEN DINH VINH, EIU

54

The first data column indicates whether the class itself has access to the member defined by the access level. As you can see, a class always has access to its own members.

The second column indicates whether classes in the same package as the class have access to the member.

The third column indicates whether subclasses of the class declared outside this package have access to the member.

The fourth column indicates whether all classes have access to the member.

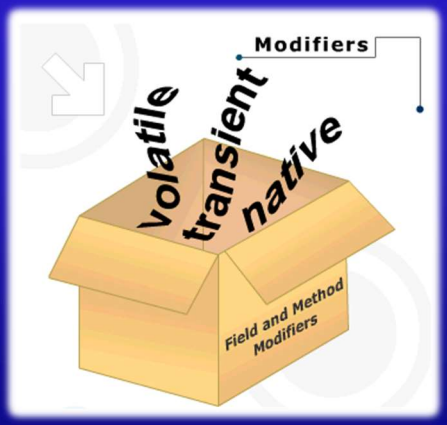
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
Protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

DR. NGUYEN DINH VINH, EIU

55

Field and Method Modifiers

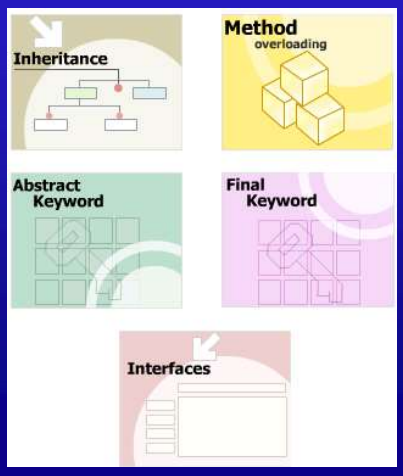
- Use to identify fields and methods that need to be declare for controlling access to users.



DR. NGUYEN DINH VINH, EIU

56

INHERITANCE, INTERFACES AND POLYMORPHISM

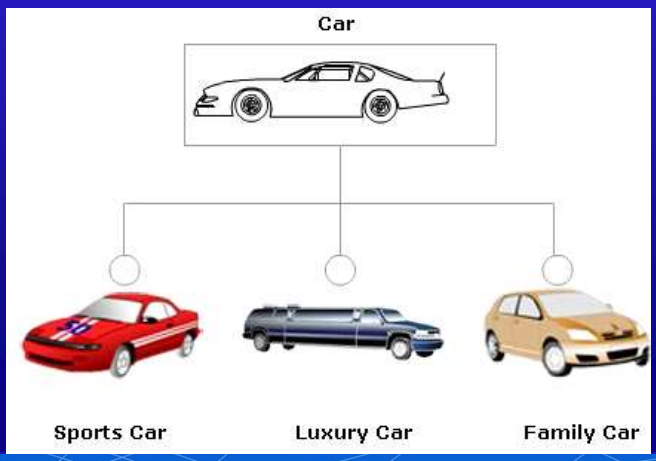


- Inheritance
- Constructor Inheritance
- Overriding Methods
- Overloading of methods
- "abstract" class
- Using "final" keyword
- Interfaces
- Polymorphism

DR. NGUYEN DINH VINH, EIU

72

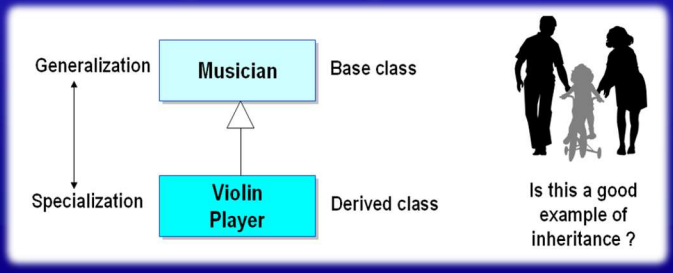
WHAT IS INHERITANCE?



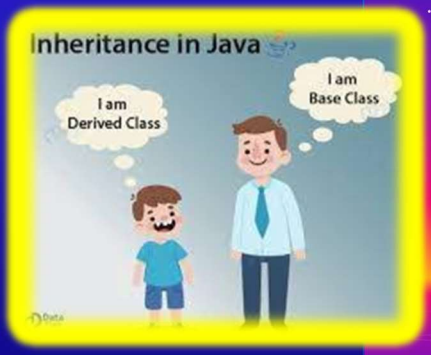
73

Basic concepts of Inheritance

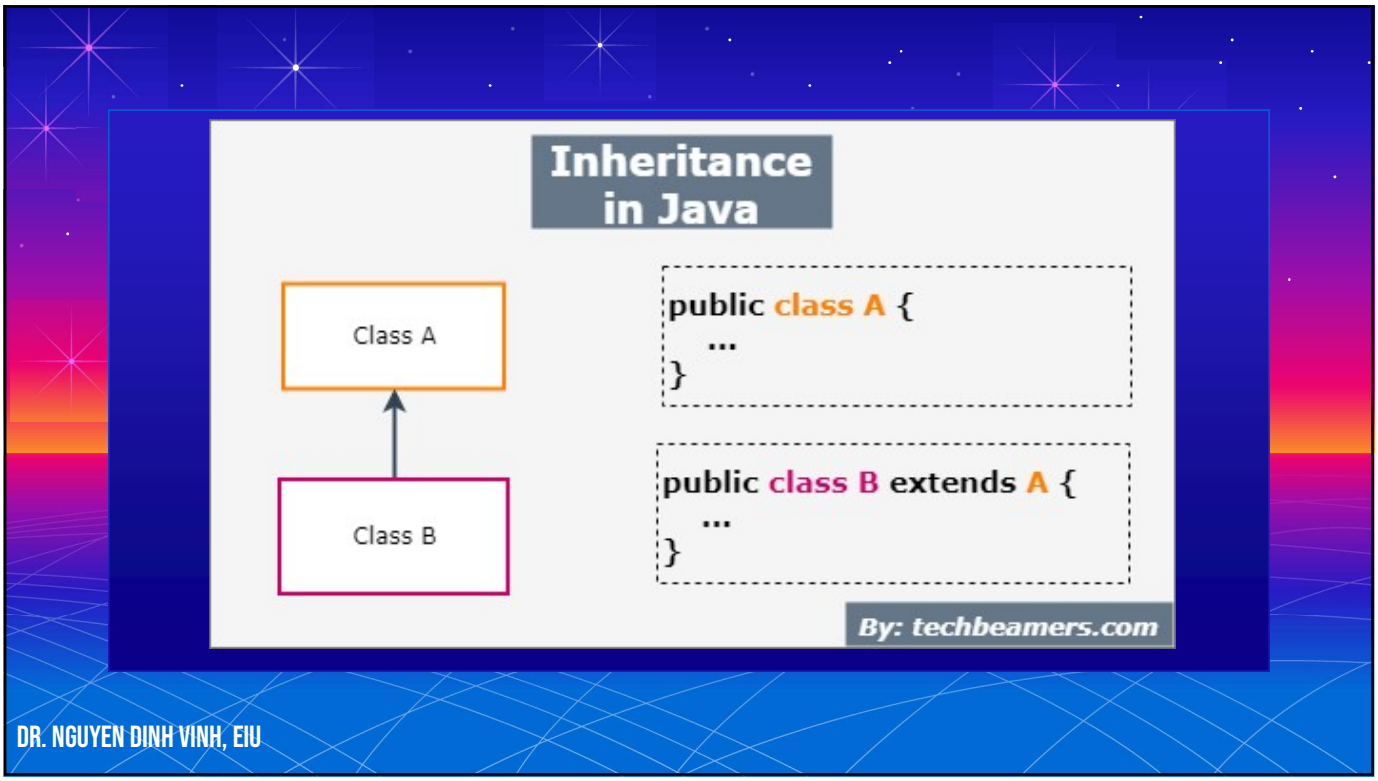
- **Superclass**
  - The class from which the subclass is derived is called a
    - Base class or parent class.
- **Subclass**
  - Class that is derived from another class
    - Derived class, extended class, or child class



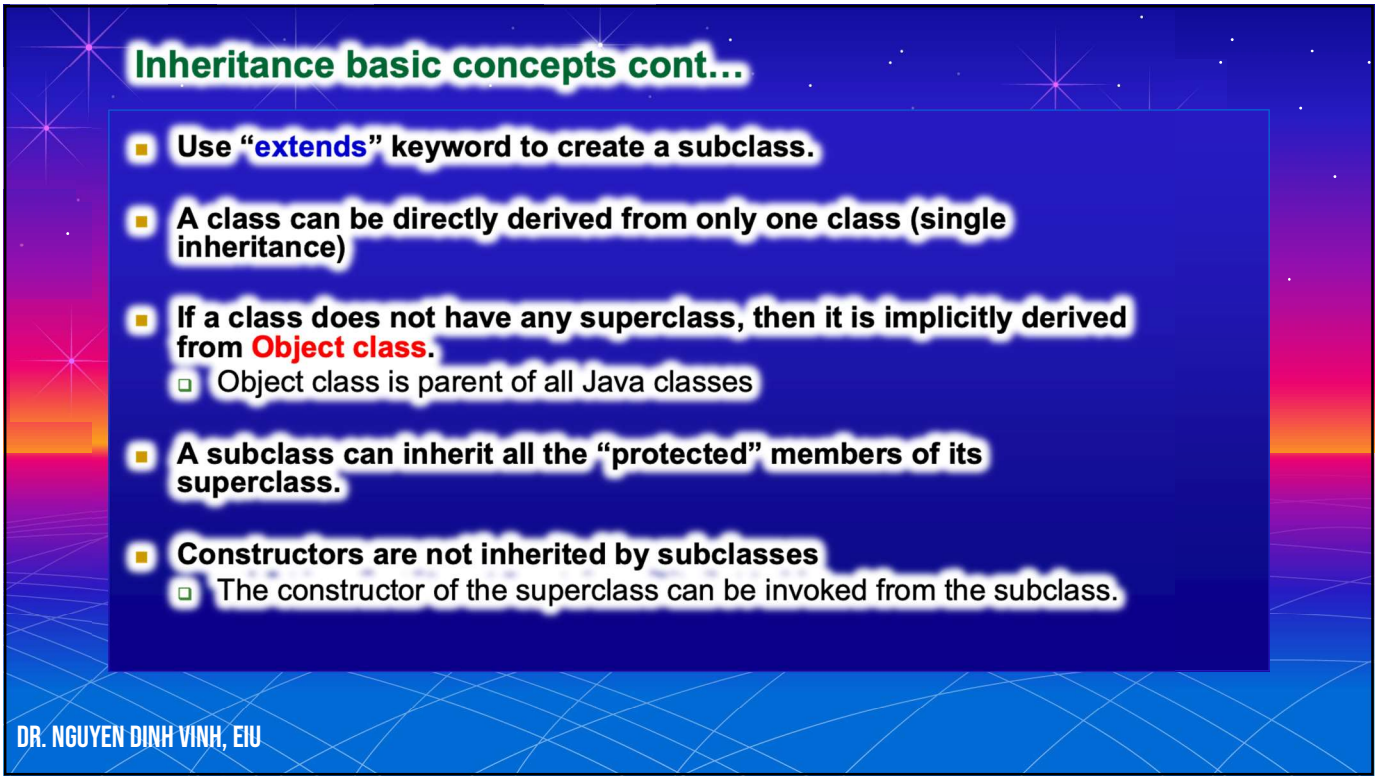
Is this a good example of inheritance ?



74


Inheritance in Java
Class A
Class B
public class A {
...
}
public class B extends A {
...
}
By: techbeamers.com
DR. NGUYEN DINH VINH, EIU

75


Inheritance basic concepts cont…
Use "extends" keyword to create a subclass.
A class can be directly derived from only one class (single inheritance)
If a class does not have any superclass, then it is implicitly derived from Object class.
Object class is parent of all Java classes
A subclass can inherit all the "protected" members of its superclass.
Constructors are not inherited by subclasses
The constructor of the superclass can be invoked from the subclass.
DR. NGUYEN DINH VINH, EIU

76

```
class Calculator {
    int add(int a , int b)
    {
        return a + b;
    }

    int sub(int a , int b)
    {
        return a - b;
    }
}
```

```
public class AdvancedCalculator extends Calculator {
    int mult(int a , int b)
    {
        return a * b;
    }

    int div(int a , int b)
    {
        return a / b;
    }
public static void main(String args[])
{
    AdvancedCalculator cal = new AdvancedCalculator();

    System.out.println( cal.add(1, 2) );
    System.out.println( cal.sub(1, 2) );
    System.out.println( cal.mult(1, 2) );
    System.out.println( cal.div(1, 2) );
```


77

```
/**
 *
 * @author dinhvinhnguyen
 */
class Parent {

    int num = 1;
    int a;

    Parent(int a)
    {
        this.a = a;
    }

    void disp( ) {
        System.out.println("Number from Parent class " + num);
    }

    void getA( ) {
        System.out.println("Value of a " + a);
    }
}
```

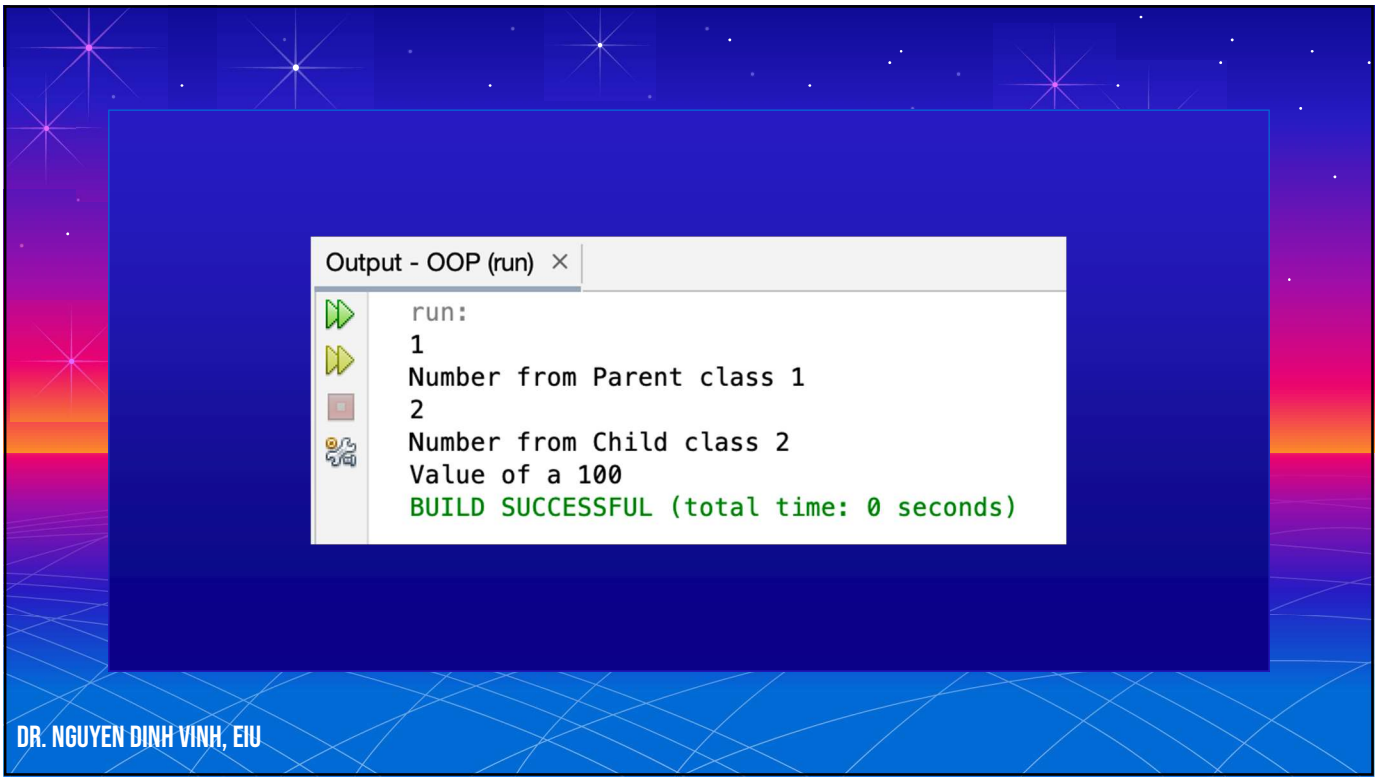

78

```
 *
 * @author dinhvinhnguyen
 */
public class Children extends Parent {

    int num = 2;

    Children(int a) {
        super( a );
    }

    void disp( ) {
        System.out.println("Number from Child class " + num);
    }

    void methodToCallSuperKeyword( ) {
        System.out.println(super.num);
        super.disp();
    }

}
```

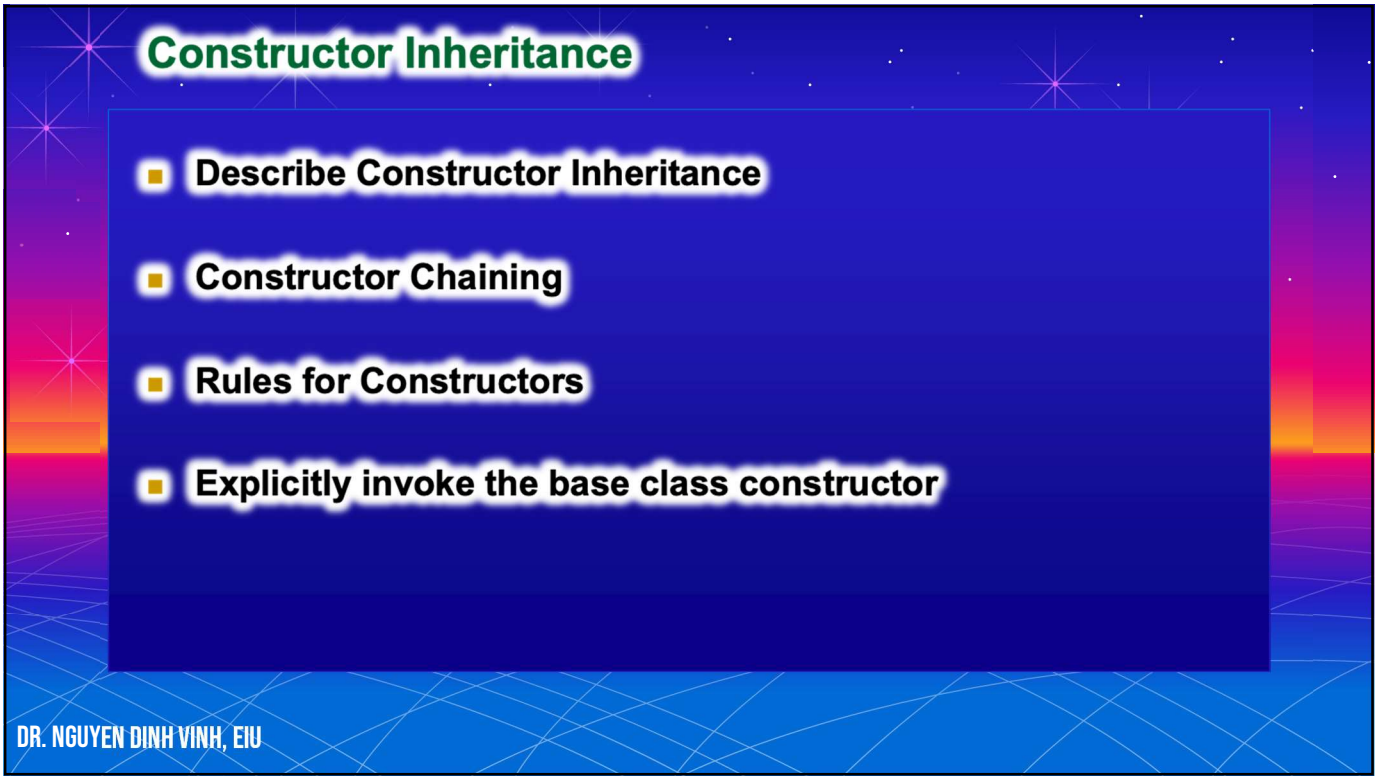

79

```
/**
 *
 * @author dinhvinhnguyen
 */
public class Test {
    public static void main(String args[]) {

        Children child = new Children(100);

        child.methodToCallSuperKeyword();
        System.out.println(child.num);

        child.disp();
        child.getA();
    }
}
```


80


Output - OOP (run)
run:
1
Number from Parent class 1
2
Number from Child class 2
Value of a 100
BUILD SUCCESSFUL (total time: 0 seconds)
DR. NGUYEN DINH VINH, EIU

81


Constructor Inheritance
Describe Constructor Inheritance
Constructor Chaining
Rules for Constructors
Explicitly invoke the base class constructor
DR. NGUYEN DINH VINH, EIU

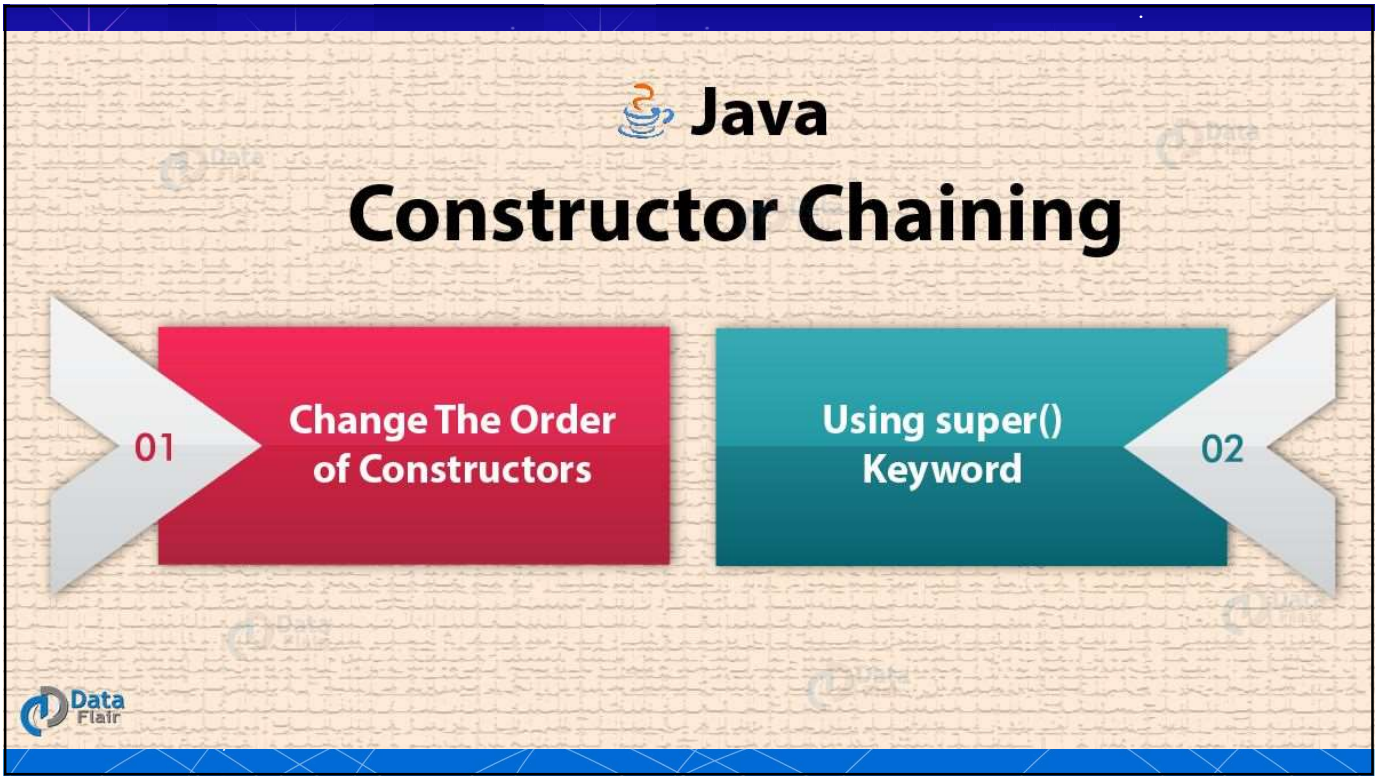
82

Constructor Inheritance

- In Java, cannot inherit constructors like inherit methods.
- The instance of the derived class will always first invoke the constructor of the base class followed by the constructor of the derived class.
- Can explicitly invoke the base class constructor by using the super keyword in the derived class constructor declaration

DR. NGUYEN DINH VINH, EIU

83



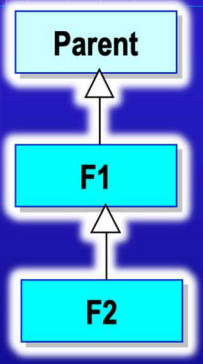
84

Constructor Chaining

```
public class Parent
{
    public Parent()
    {
        System.out.println("Invoke parent default constructor");
    }
}
```

```
public class F1 extends Parent
{
    public F1()
    {
        System.out.println("Invoke F1 default constructor");
    }
}
```

```
public class F2 extends F1
{
    public F2()
    {
        System.out.println("Invoke F2 default constructor");
    }
}
```



85

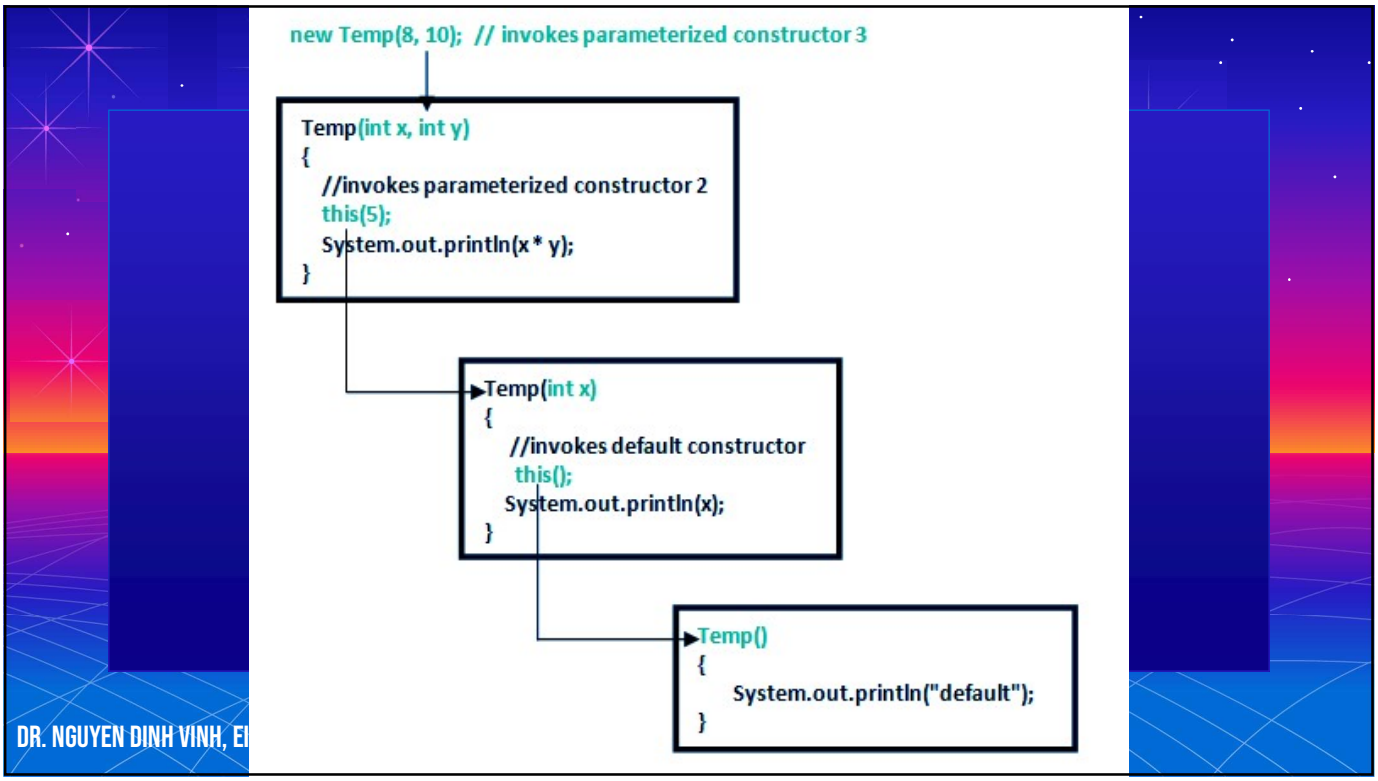
Constructor Chaining

```
public static void main(String [] args)
{
    new F2();
}
```

1. Object
2. Parent() call super()
3. F1() call super ()
4. F2() call super()
5. Main() call new F2()

- The instance of the derived class will always first invoke the constructor of the base class followed by the constructor of the derived class

86



87

Rules for Constructors *(MUST be clear and remember)*

- A default constructor will be automatically generated by the compiler if no constructor is specified within the class.
- The default constructor is ALWAYS a no-arg constructor.
- If there is a constructor defined in the class, the default constructor is no longer used.
- If you don't explicitly call a base class constructor in a derived class constructor, the compiler attempts to silently insert a call to the base class's default constructor before executing the code in the derived class constructor

DR. NGUYEN DINH VINH, EIU

88

Can you see any bug?

```
public class Parent
{
    private int a;

    public Parent(int value)
    {
        a = value;
        System.out.println("Invoke parent parameter constructor");
    }
}
```

```
public class F1 extends Parent
{
    public F1()
    {
        System.out.println("Invoke F1 default constructor");
    }
}
```

DR. NGUYEN DINH VINH, EIU

90

How to fixed?

```
public class Parent
{
    private int a;

    public Parent()
    {
        a = 0;
        System.out.println("Invoke parent default constructor");
    }

    public Parent(int value)
    {
        a = value;
        System.out.println("Invoke parent parameter constructor");
    }
}
```

```
public class F1 extends Parent
{
    public F1()
    {
        System.out.println("Invoke F1 default constructor");
    }
}
```

DR. NGUYEN DINH VI

91

Explicitly invoke the base class constructor

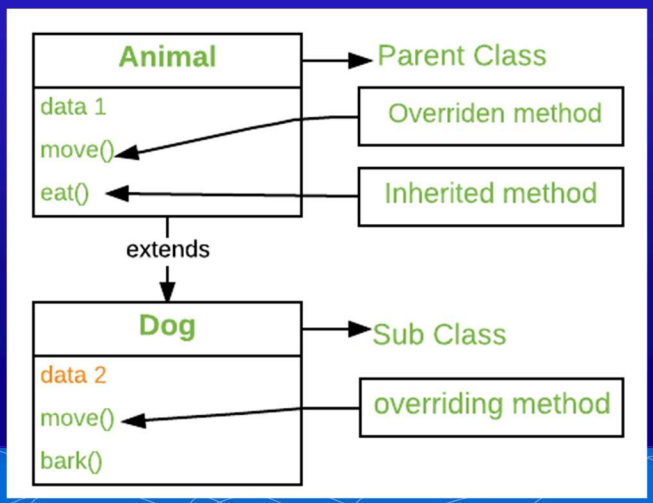
```
public class Parent
{
    private int a;

    public Parent(int value)
    {
        a = value;
        System.out.println("Invoke parent parameter constructor");
    }
}
```

```
public class F1 extends Parent
{
    public F1(int value)
    {
        super (value);
        System.out.println("Invoke F1 default constructor");
    }
}
```

DR. NGUYEN DI

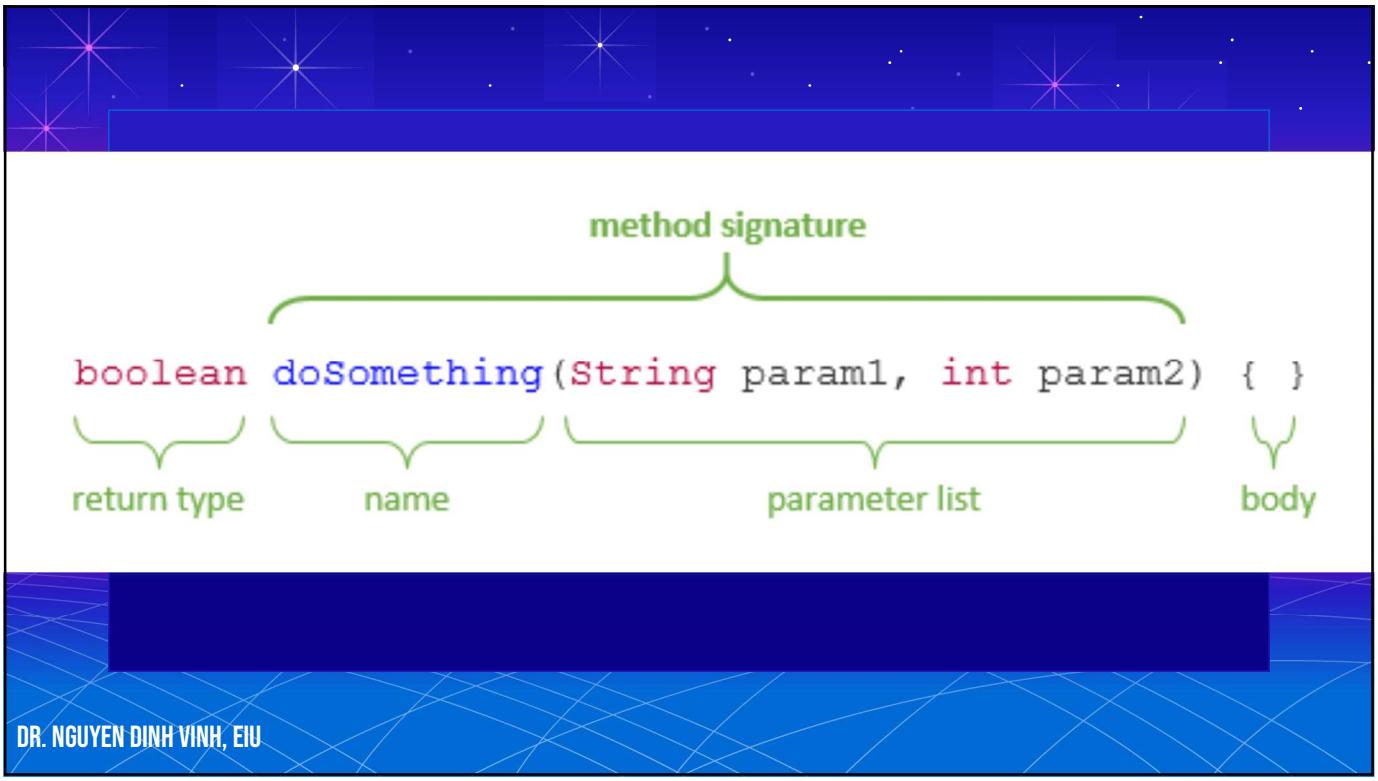
92

OVERRIDING METHODS

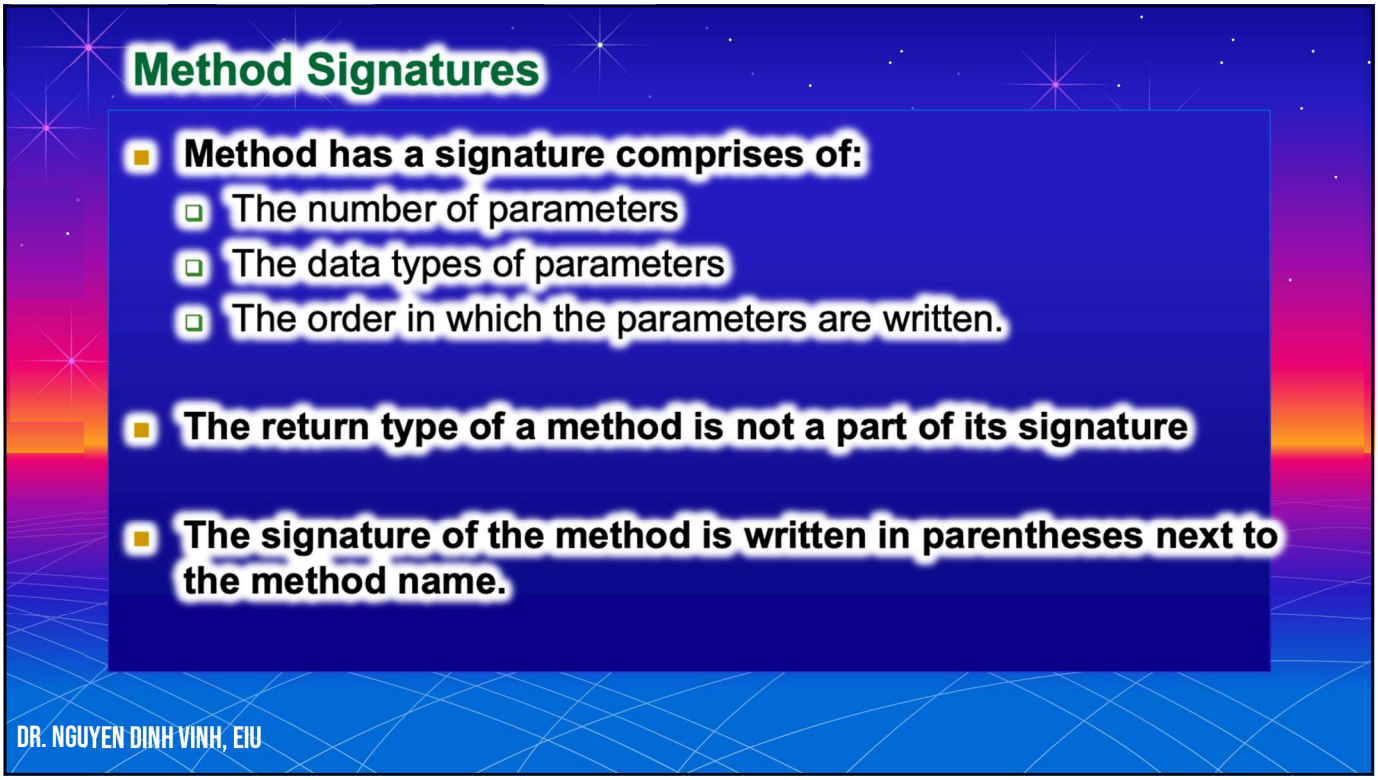


DR. NGUYEN DINH VINH, EIU

93


method signature
boolean doSomething(String param1, int param2) { }
return type
name
parameter list
body
DR. NGUYEN DINH VINH, EIU

94


Method Signatures
Method has a signature comprises of:
The number of parameters
The data types of parameters
The order in which the parameters are written.
The return type of a method is not a part of its signature
The signature of the method is written in parentheses next to the method name.
DR. NGUYEN DINH VINH, EIU

95

Overriding Methods

- Subclass define new method with the same signature as the superclass method.
- Overridden method cannot have a weaker "access specifier" than the "access specifier" of the method it overrides.
  - Superclass: protected
  - Subclass: public

  - Superclass: public
  - Subclass: protected

DR. NGUYEN DINH VINH, EIU

96

Rules for Java Method Overriding

STEP 01 — Method must have same name as in the parent class

STEP 02 — Method must have same parameter as in the parent class.

STEP 03 — There must be IS-A relationship (inheritance).

DR. NGUYEN DINH

97

Example of Overriding Methods

```
public class Parent
{
    public void printInfor()
    {
        System.out.println("Calling printInfor()method of Parent");
    }
}
```

```
public class F1 extends Parent
{
    public void printInfor()
    {
        System.out.println("Calling printInfor()method of F1");
    }
}
```


98

"super" keyword

- Use to access superclass's members and constructors from subclass

99

Example of Using "super" keyword

```
public class Parent
{
    public void printInfor()
    {
        System.out.println("Calling printInfor()method of Parent");
    }
}

public class F1 extends Parent
{
    public void printInfor()
    {
        super.printInfor();
        System.out.println("Calling printInfor()method of F1");
    }
}
```

```
public static void main (String [] args)
{
    F1 f1 = new F1();

    f1.printInfor();
}
```

```
Console
<terminated> Parent [Java Application] C:\Program Files\Java\jdk1.5.0_11\bin\j
Calling printInfor()method of Parent
Calling printInfor()method of F1
```


100

Method Overloading

```
public static int sum(int x, int y) {
    return x + y;
}

public static int sum(int x, int y, int z) {
    return x + y + z;
}
```


101

Overloading Methods

- Declaring more than one method with the same method name but different signatures
- Constructor overloading
  - Allows multiple ways of creating instances

102

Example of Overloading Methods

```
public class OverloadClass
{
    private int a;

    public OverloadClass()
    {
        a= 0;
    }

    public OverloadClass(int value)
    {
        a = value;
    }

    public int add(int valOne, int valTwo)
    {
        return (valOne + valTwo);
    }

    public float add(float valOne, float valTwo
    {
        return (valOne + valTwo);
    }
}
```


103

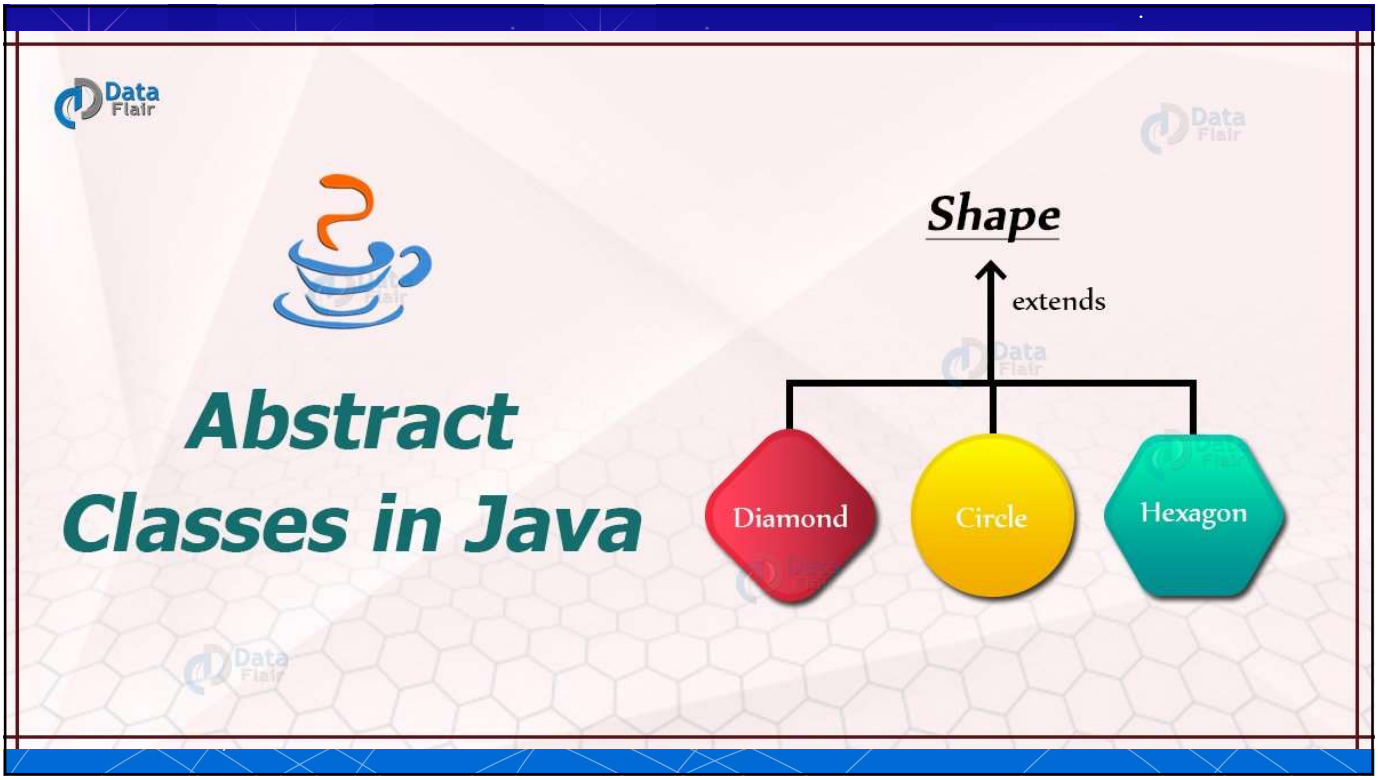
"this" keyword

- Refer to the current object of the class.
- Used to resolve conflicts between variables having same names and to pass the current object as a parameter
- Cannot use the "this" keyword with static variables and methods,

```
public class Rectangle
{
    private float length;
    private float width;

    public Rectangle()
    {
        this.length = 0;
        this.width = 0;
    }

    public Rectangle(float length, float width)
    {
        this.length = length;
        this.width = width;
    }

    public float area()
    {
        return (length * width);
    }
}
```

DR. NGUYEN DINH VINH, EIU

104



105

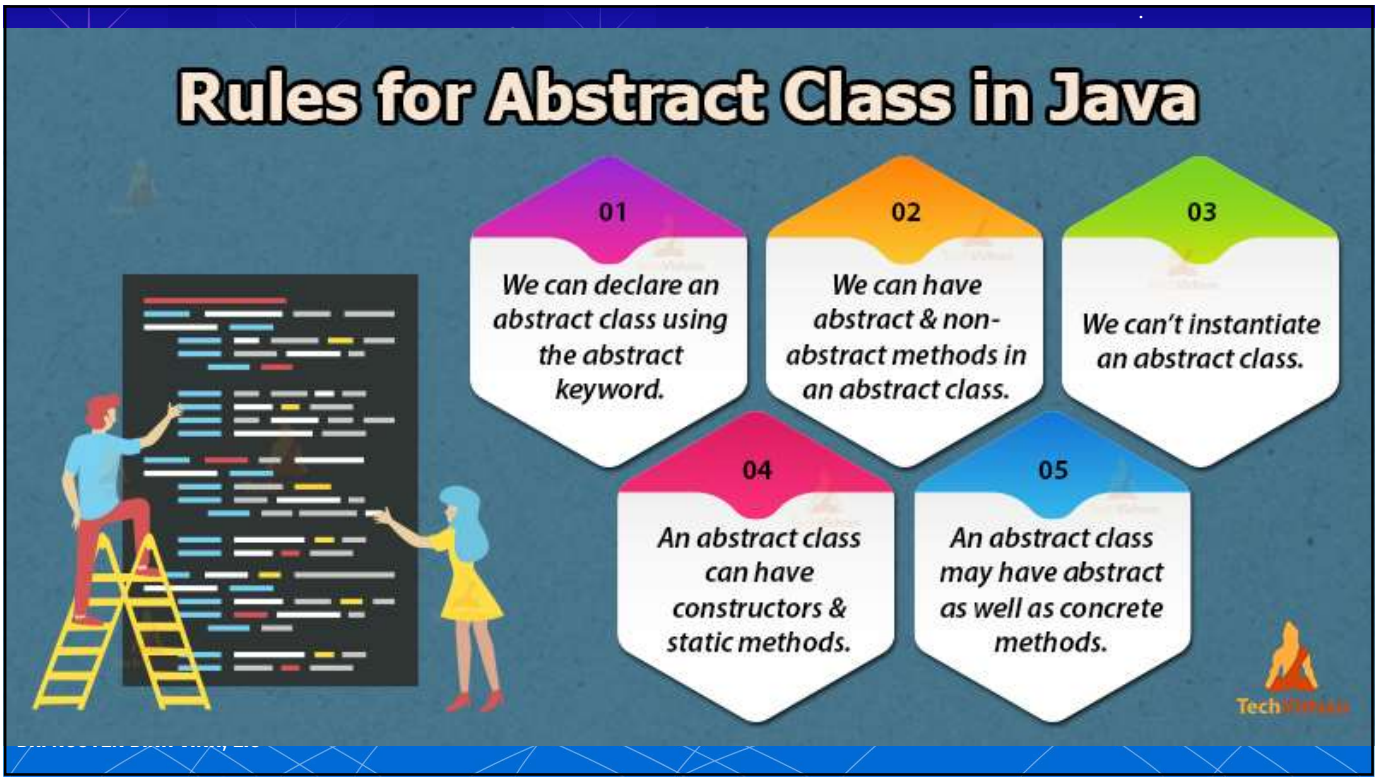
"abstract" Classes

- To serve as a framework that provides certain behavior for other classes.
- Contain zero or more abstract methods
- Cannot be instantiated.
- Can be inherited.
- The subclass must implement "abstract methods" declared in base class
  - Otherwise it must be declared as abstract.
- Declare an abstract class by using the keyword abstract precede class keyword

106

```
abstract class Person {
    String name;
   abstract String getDetails();

   }
class Student extends Person{
     int rollNo;

}

class Teacher extends Person{

}
class Employee extends Person{
```


107



108

"abstract" Methods

- **Method has only declaration and no implementation**
  - Is prefixed with the "abstract" keyword.
  - The declaration does not contain any braces and is terminated by a semicolon.
- **An abstract method is only a contract that the subclass will provide its implementation**
- **If a class includes abstract methods, the class itself must be declared abstract**

109

```
abstract class Animals{
    private String name;

    // All kind of animals eat food to make this common to all animals
    public void eat(){
        System.out.println(" Eating ..........");
    }

    // The animals make different sounds. They will provide their own implementation
    abstract void sound();
}

class Cat extends Animals{

    @Override
    void sound() {
        System.out.println("Meoww Meoww ........");
    }
}
```

abstract class

abstract method

abstract method implementation

DR. NGUYEN DI

110

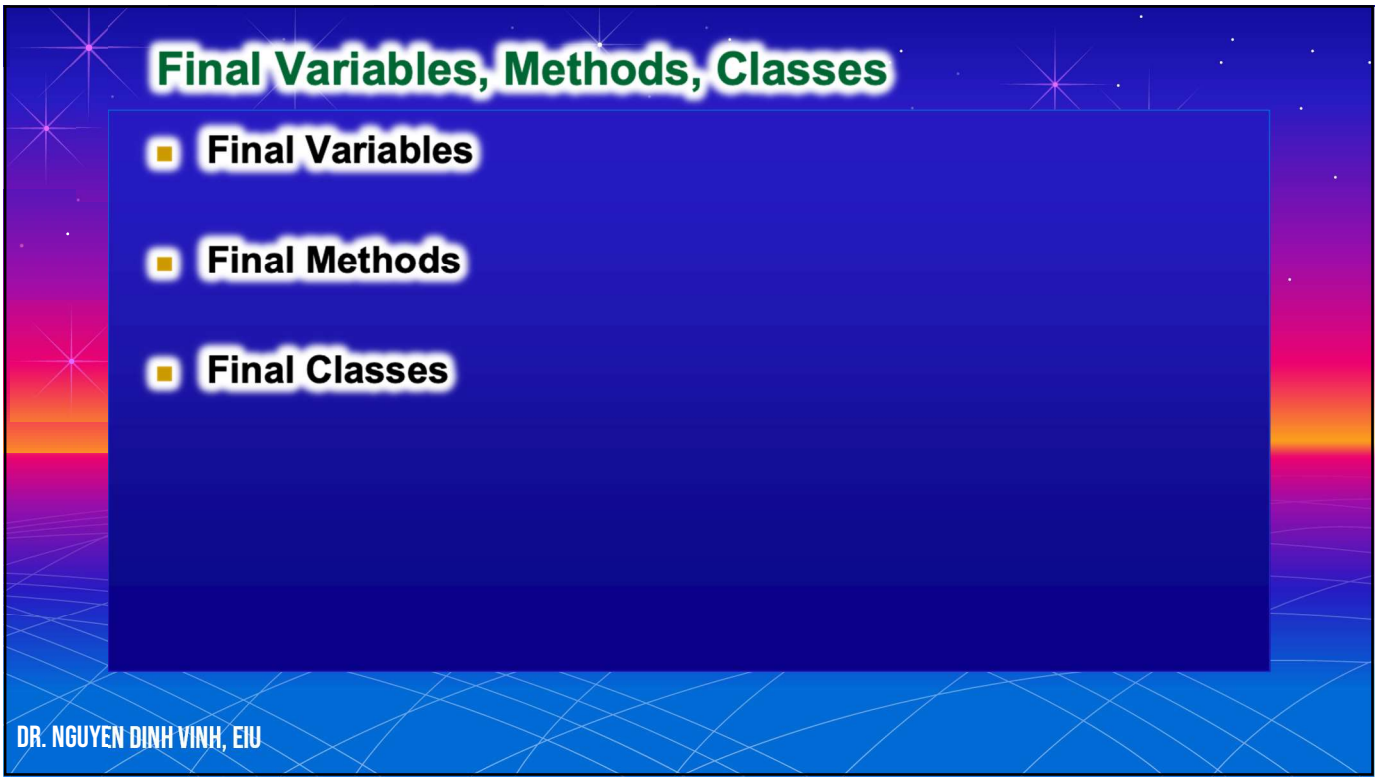
Example of "abstract" Classes and "abstract" Methods

```
public abstract class Person
{
    protected int id;
    protected String name;

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

    public abstract void displayInfor();
}
```

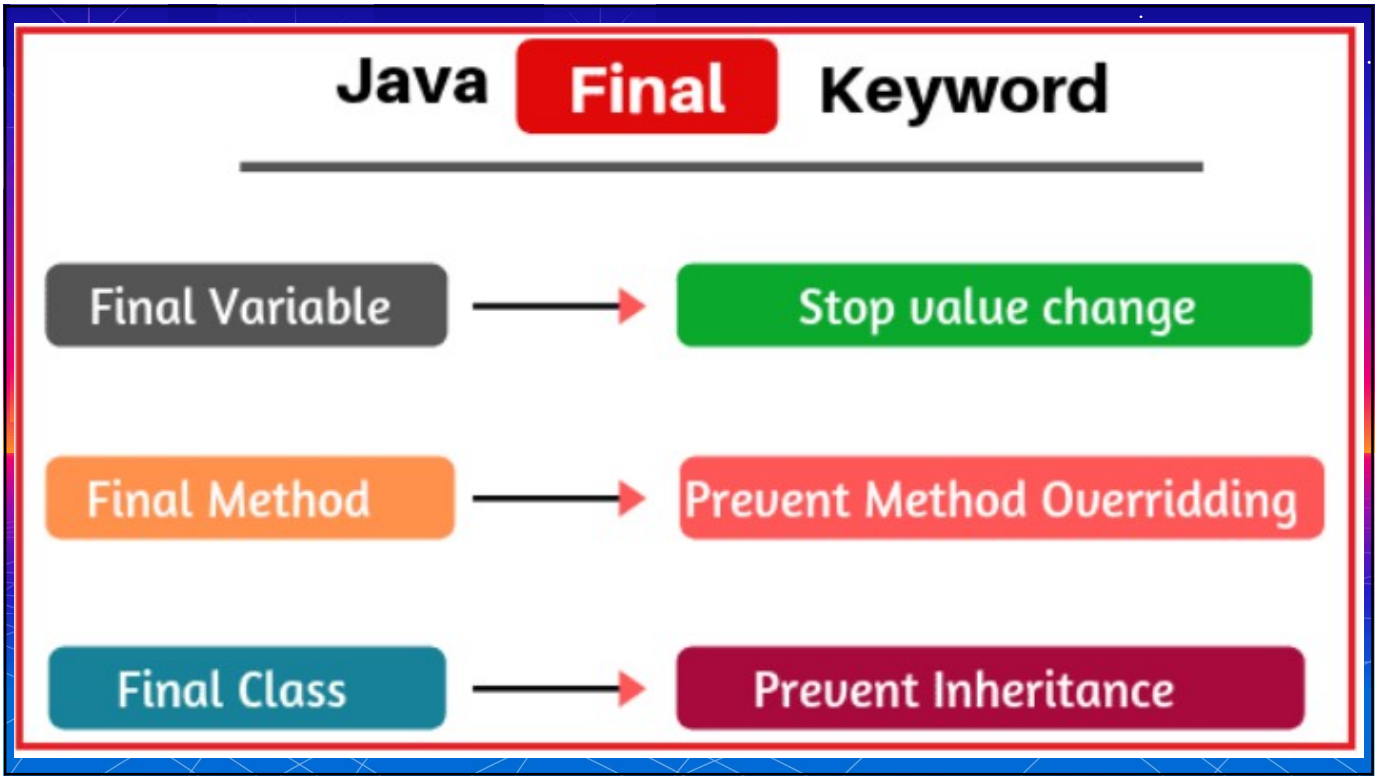
```
public class Student extends Person
{
    public Student(int id, String name)
    {
        super(id, name);
    }

    public void displayInfor()
    {
        System.out.println("Student id: " + super.id);
        System.out.println("Student name: " + super.name);
    }
}
```

DR. NGUYEN DI

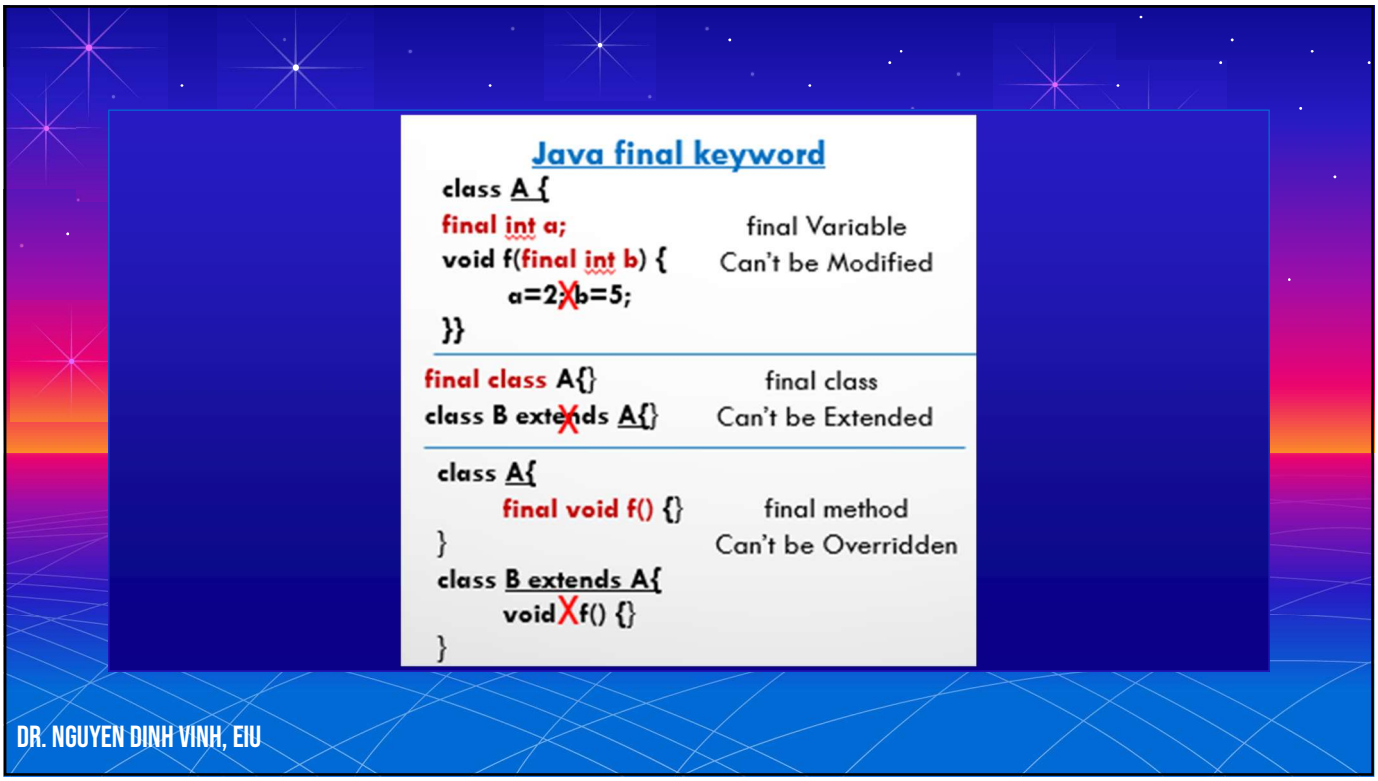
111


Final Variables, Methods, Classes
Final Variables
Final Methods
Final Classes
DR. NGUYEN DINH VINH, EIU

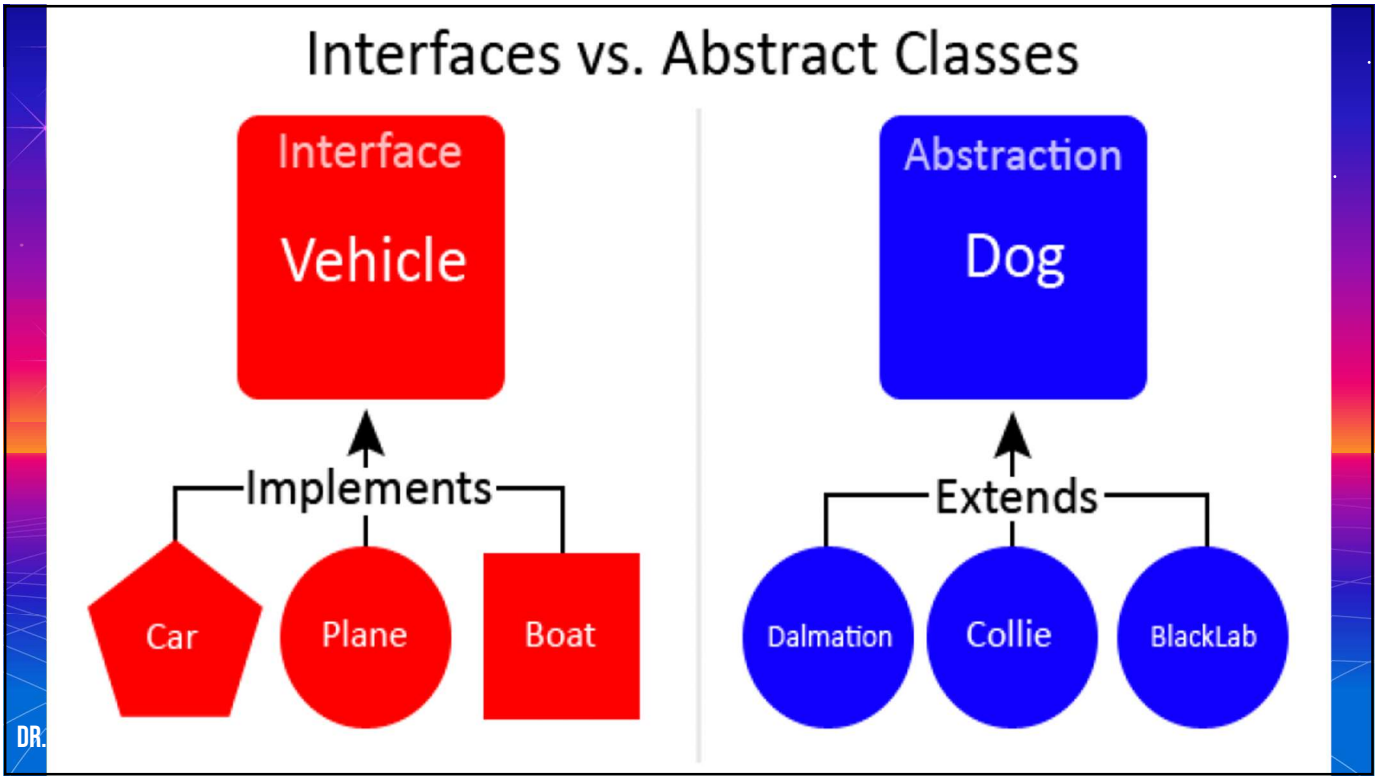
112


Java Final Keyword
Final Variable
Stop value change
Final Method
Prevent Method Overridding
Final Class
Prevent Inheritance

113


Java final keyword
class A {
final int a;
void f(final int b) {
a=2;b=5;
}}
final Variable
Can't be Modified
final class A{}
class B extends A{}
final class
Can't be Extended
class A{
final void f() {}
}
class B extends A{
void f() {}
}
final method
Can't be Overridden
DR. NGUYEN DINH VINH, EIU

114


Interfaces vs. Abstract Classes
Interface
Vehicle
Implements
Car
Plane
Boat
Abstraction
Dog
Extends
Dalmation
Collie
BlackLab
DR.

115

What is an Interface?

- An interface is defined as a reference type
- Has only final variables, abstract methods signatures.
  - Interface Methods Do Not Contain Method Bodies
- Cannot be instantiated.
- Can only be inherited by classes or other interfaces.
- A class that implements an interface is *required to provide implementations for all the methods* of the interface or else should be declared abstract.

116

Example of Interface

```
public interface UserDao {

    public final String TABLE ="User";

    public void create(User user);

    public void delete(int id);

    public void update(int id, User user);

}
```

```
public class UserDaoImpl implements UserDao {

    public void create(User user) {
        // ...
    }

    public void delete(int id) {
        // ...
    }

    public void update(int id, User user) {
        // ....
    }
}
```


117

Example of Implementing Multiple Interfaces

```
public interface Quackable
{
    public void quack();
}
```

```
public interface Flyable
{
    public void fly();
}
```

```
public class RedheadDuck implements Quackable, Flyable
{
    public void quack()
    {
    }

    public void fly()
    {
    }
}
```

DR. NGU

118

Abstract Class

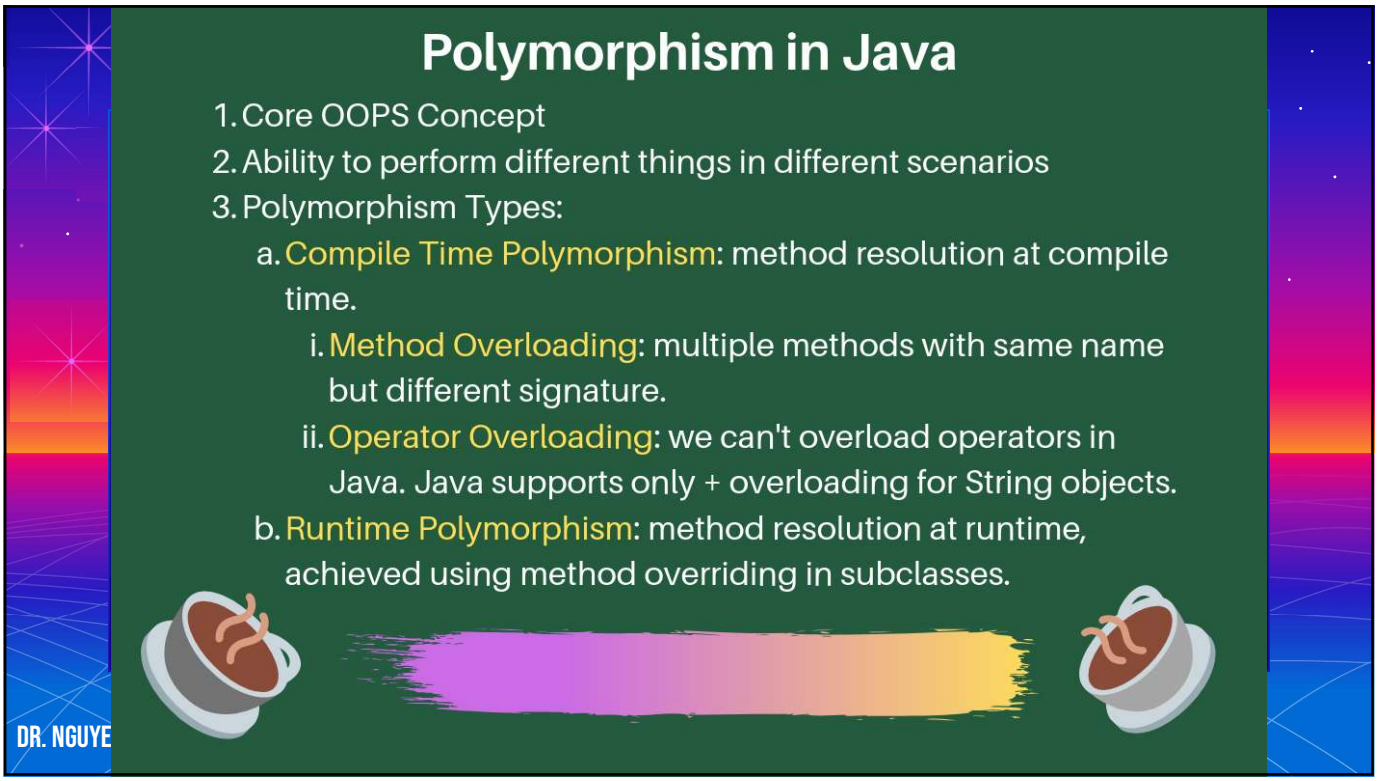
1. *abstract* keyword
2. Subclasses *extends* abstract class
3. Abstract class can have implemented methods and 0 or more abstract methods
4. We can extend only one abstract class

Interface

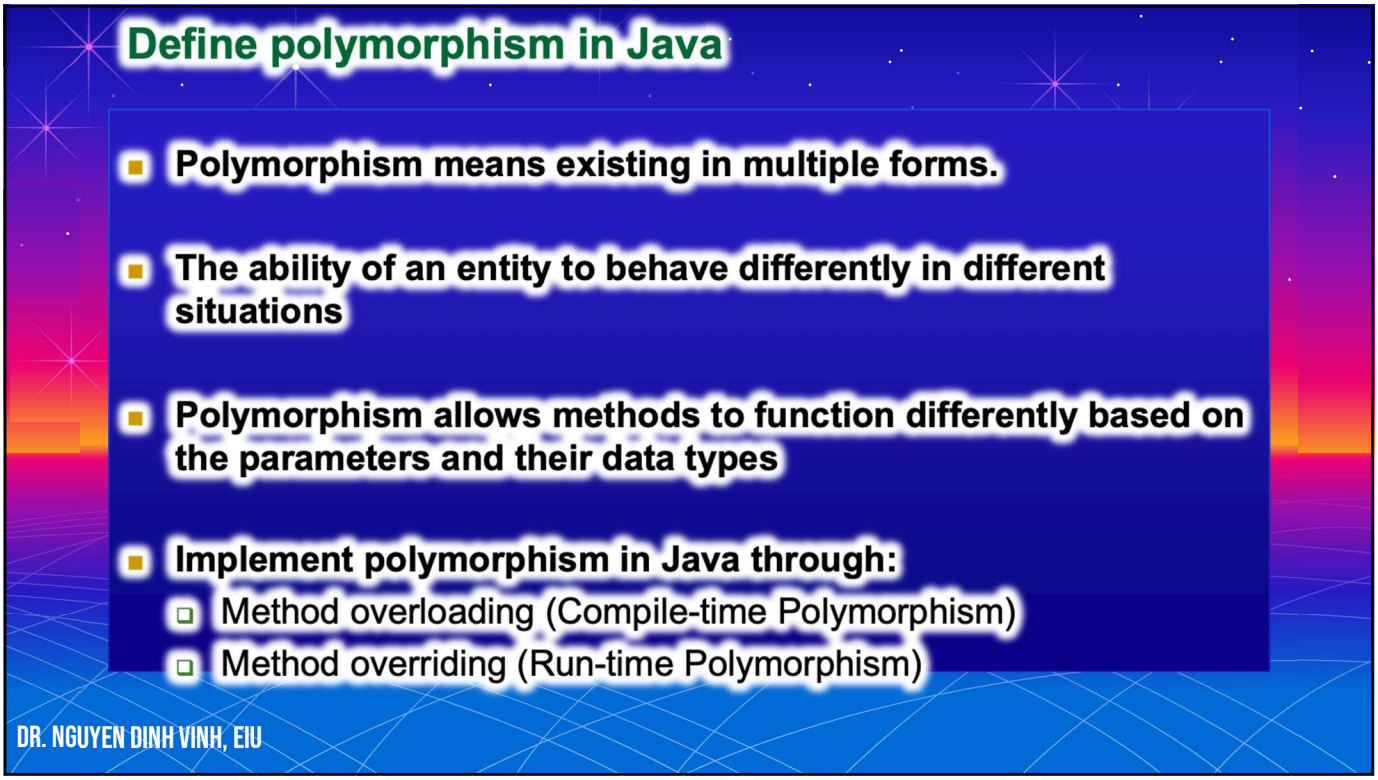
1. *interface* keyword
2. Subclasses *implements* interfaces
3. Java 8 onwards, Interfaces can have default and static methods
4. We can implement multiple interfaces

DR. NGUYE

119


Polymorphism in Java
1. Core OOPS Concept
2. Ability to perform different things in different scenarios
3. Polymorphism Types:
a. Compile Time Polymorphism: method resolution at compile time.
i. Method Overloading: multiple methods with same name but different signature.
ii. Operator Overloading: we can't overload operators in Java. Java supports only + overloading for String objects.
b. Runtime Polymorphism: method resolution at runtime, achieved using method overriding in subclasses.
DR. NGUYE

120


Define polymorphism in Java
Polymorphism means existing in multiple forms.
The ability of an entity to behave differently in different situations
Polymorphism allows methods to function differently based on the parameters and their data types
Implement polymorphism in Java through:
Method overloading (Compile-time Polymorphism)
Method overriding (Run-time Polymorphism)
DR. NGUYEN DINH VINH, EIU

121

Example of polymorphism in Java (Compile-Time)

```
public class A {
    public void doSomething()
    {
        System.out.println("A() do something");
    }
    public void doSomething(int a)
    {
        System.out.println("A() do something with parameter "+ a);
    }
}
public class Main {

    public static void main(String[] args) {
        A a = new A();
        a.doSomething();
        a.doSomething(5);
    }
}
```

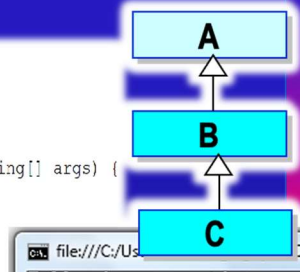
```
file:///C:/Users/kimcong.nguyen/Documents/Visual Studio 2008/Projects/
A() do something
A() do something with parameter 5
```


122

Example of polymorphism in Java (Run-Time)

```
public class A {
    public void doSomething()
    {
        System.out.println("A() do something");
    }
}
public class B extends A{
    public void doSomething()
    {
        System.out.println("B() do something");
    }
}
public class C extends B{
    public void doSomething()
    {
        System.out.println("C() do something");
    }
}
```

```
public class Main {

    public static void main(String[] args) {
        A a;
        B b=new B();
        C c=new C();

        a=b;
        a.doSomething();

        a=c;
        a.doSomething();

        b.doSomething();

        b=c;
        b.doSomething();

    }
}
```




```
B() do something
C() do something
B() do something
C() do something
```

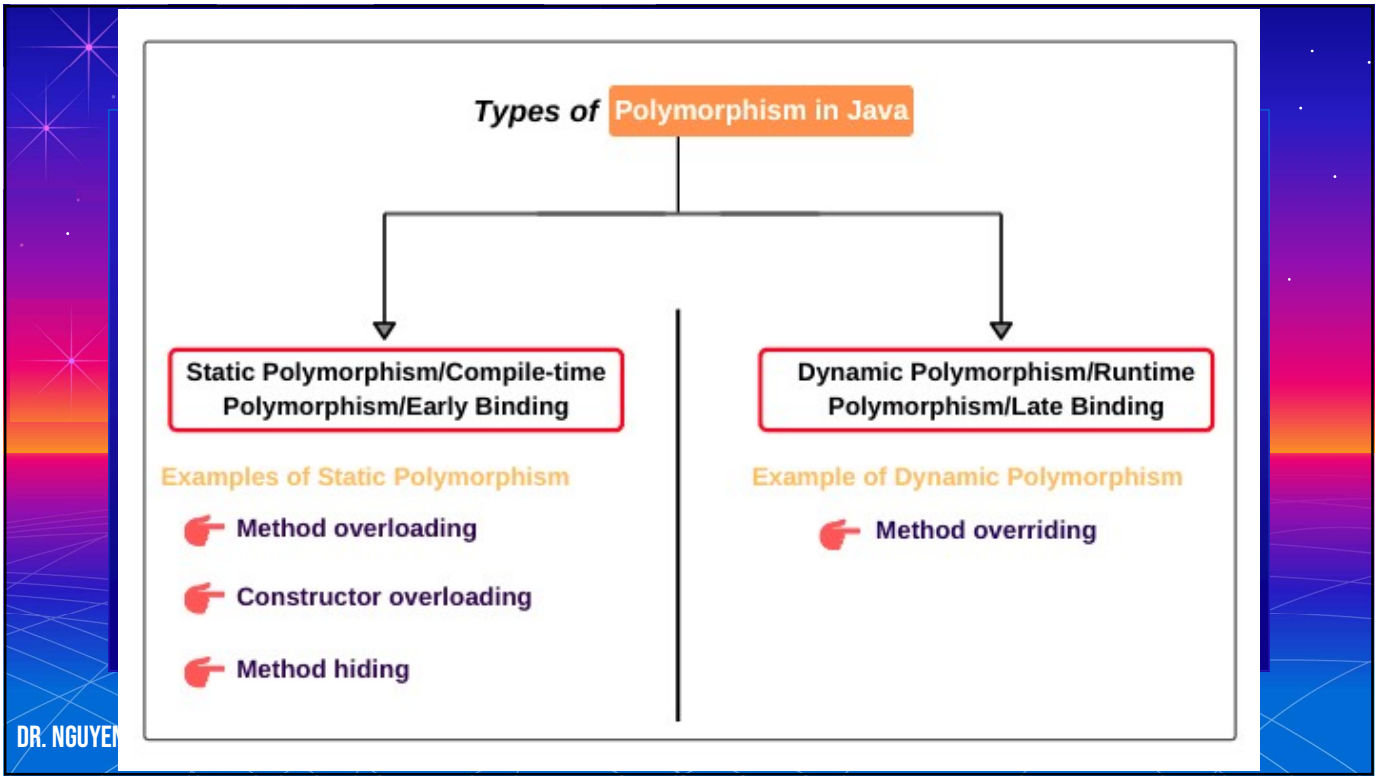

123

Compile-time and Run-time Polymorphism

- **Compile-time Polymorphism**
  - Implemented through **method overloading**
  - Executed at the compile-time
  - Referred to as **static polymorphism**
- **Run-time Polymorphism**
  - Implemented through **method overriding**
  - Executed at run-time
  - Referred to as **dynamic polymorphism**

DR. NGUYEN DINH VINH, EIU

124

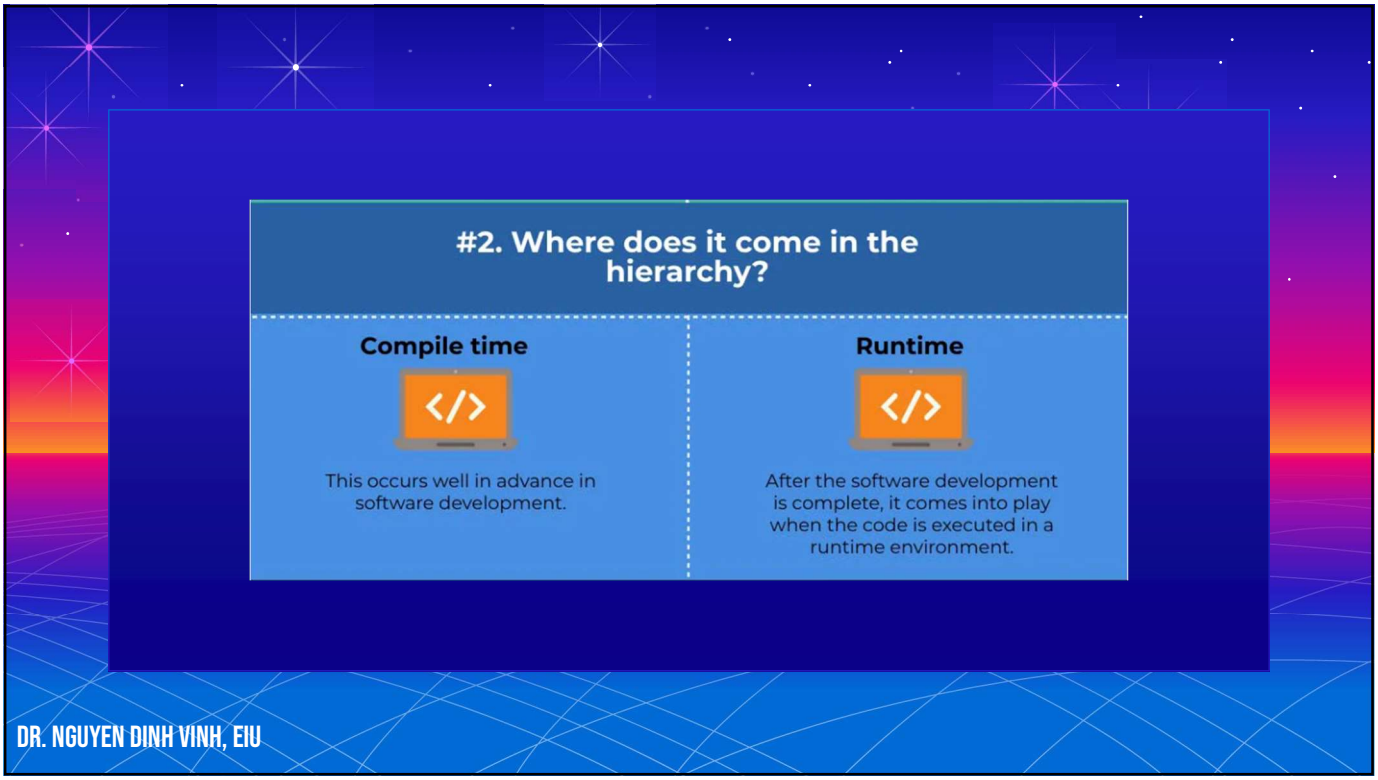


DR. NGUYEN

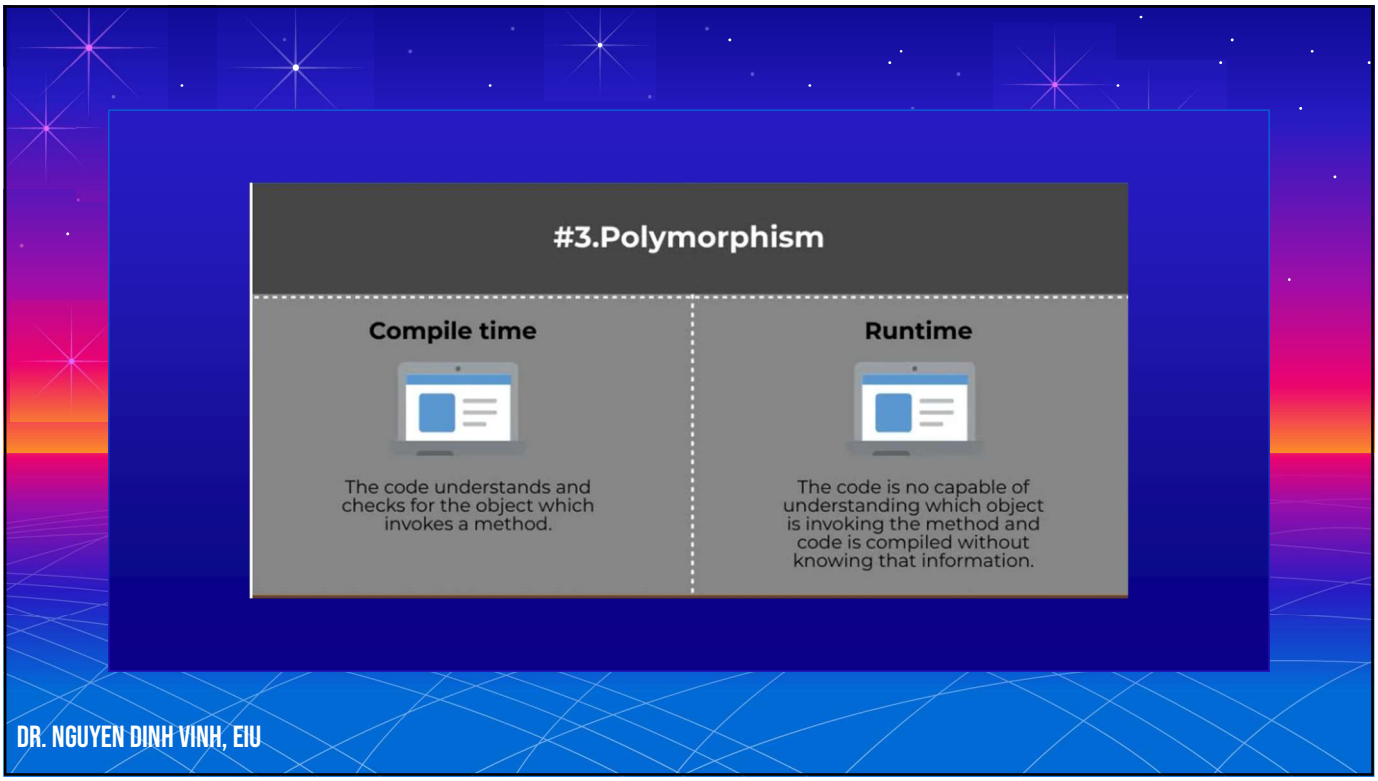
125


Compile time vs Runtime
#1. When does the time start?
Compile time
This is when the code is translated from programming language to a language that a machine understands.
Runtime
This is when a code is run in the runtime environment and starts from the time code execution starts till the point the code is stopped by the user or OS.
DR. NGUYEN DINH VINH, EIU

126


#2. Where does it come in the hierarchy?
Compile time
This occurs well in advance in software development.
Runtime
After the software development is complete, it comes into play when the code is executed in a runtime environment.
DR. NGUYEN DINH VINH, EIU

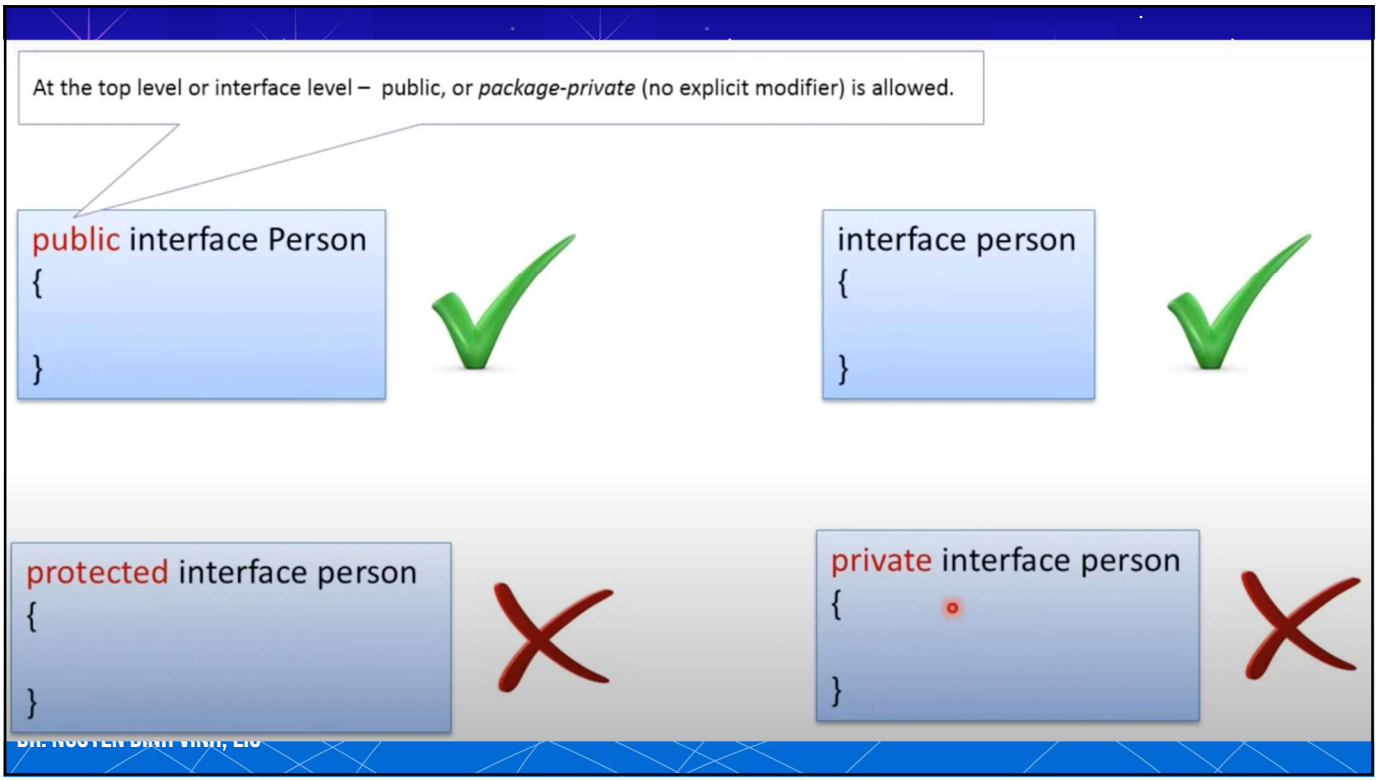
127


#3.Polymorphism
Compile time
The code understands and checks for the object which invokes a method.
Runtime
The code is no capable of understanding which object is invoking the method and code is compiled without knowing that information.
DR. NGUYEN DINH VINH, EIU

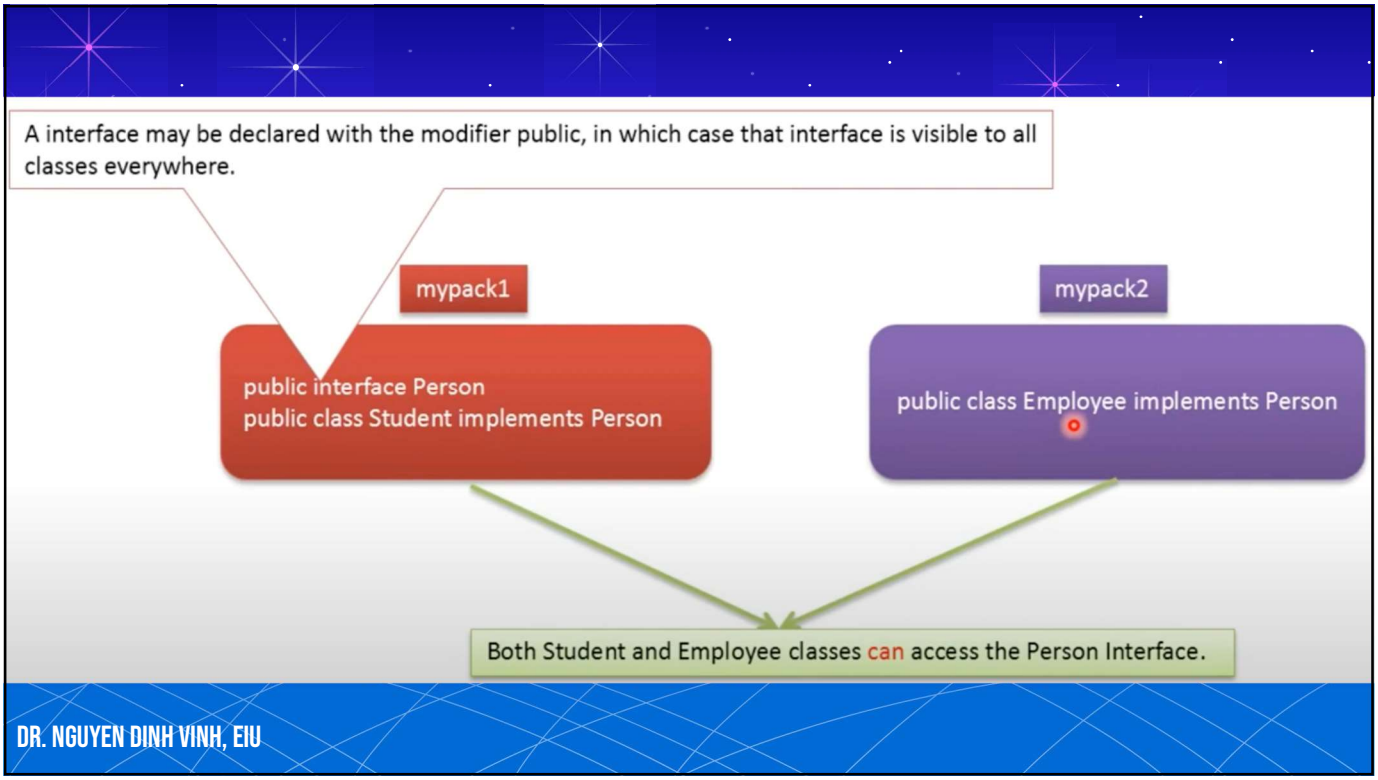
128


Bonus
DR. NGUYEN DINH VINH, EIU

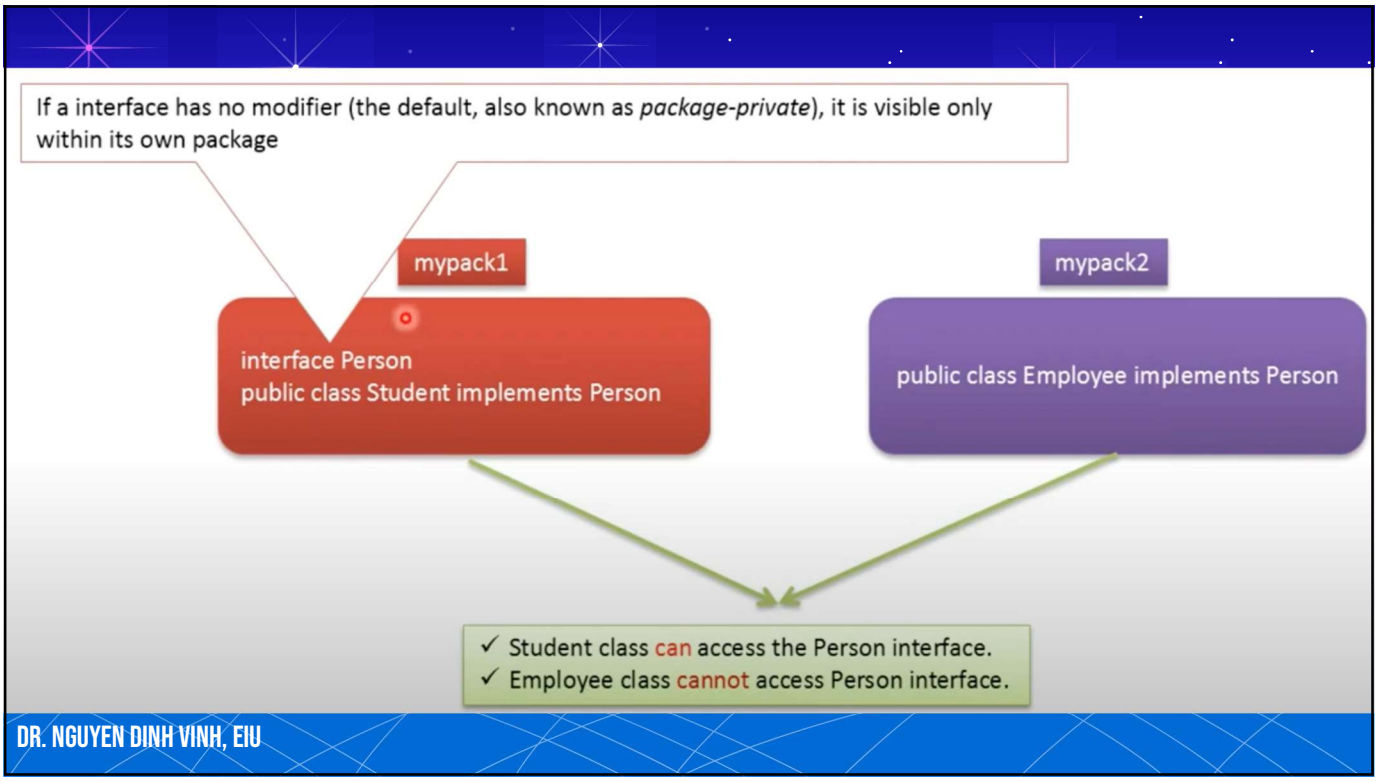
132


At the top level or interface level – public, or *package-private* (no explicit modifier) is allowed.
public interface Person
{
}
interface person
{
}
protected interface person
{
}
private interface person
{
}

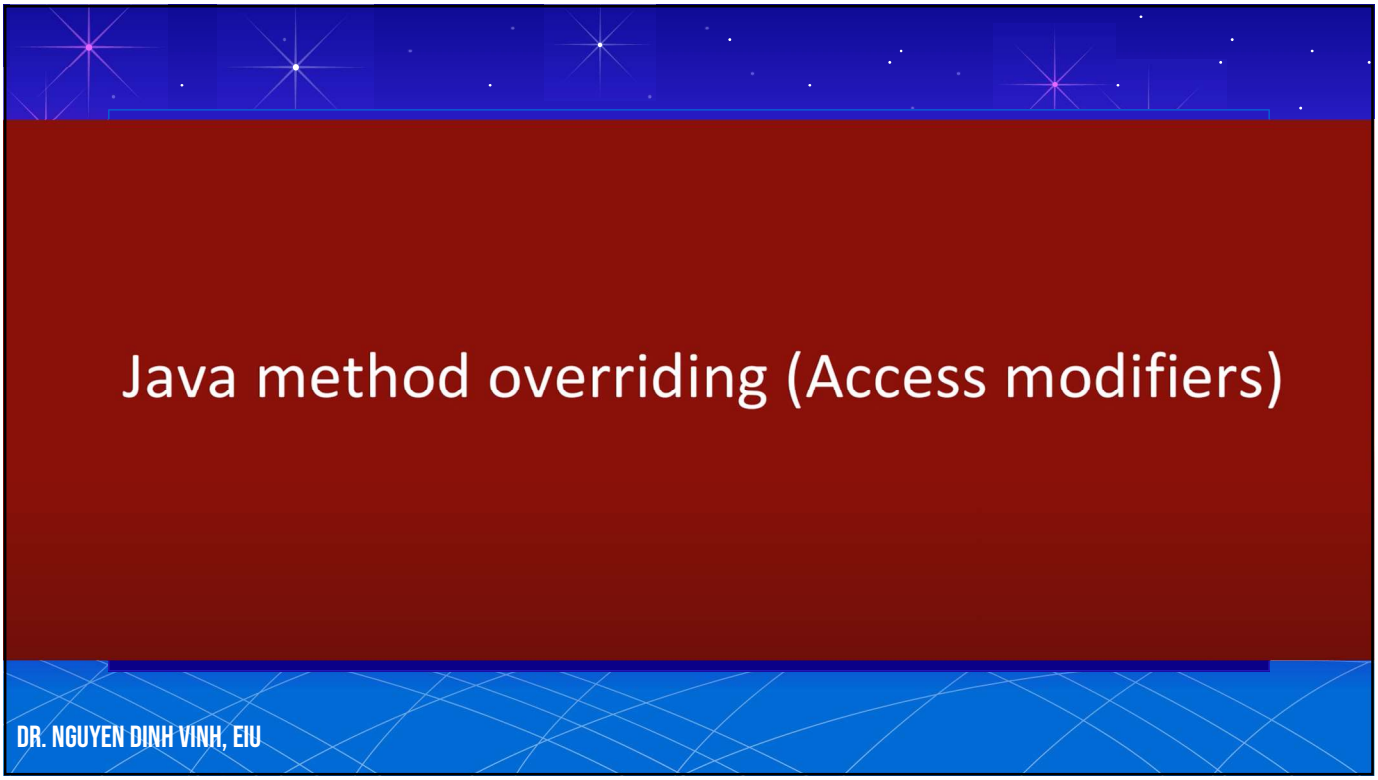
133


A interface may be declared with the modifier public, in which case that interface is visible to all classes everywhere.
mypack1
public interface Person
public class Student implements Person
mypack2
public class Employee implements Person
Both Student and Employee classes can access the Person Interface.
DR. NGUYEN DINH VINH, EIU

134


If a interface has no modifier (the default, also known as *package-private*), it is visible only within its own package
mypack1
interface Person
public class Student implements Person
mypack2
public class Employee implements Person
✓ Student class can access the Person interface.
✓ Employee class cannot access Person interface.
DR. NGUYEN DINH VINH, EIU

135


Java method overriding (Access modifiers)
DR. NGUYEN DINH VINH, EIU

136

The access specifier for an overriding method can allow more, but not less access than the overridden method. For example, a protected instance method in the superclass can be made public, but not private, in the subclass.

```
public class Animal
{
        protected void run()
        {
          System.out.println("Animal runs slowly");
        }
}

public class Tiger extends Animal
{
        protected void run()
        {
           System.out.println("Tiger runs fast.");
        }
}
```

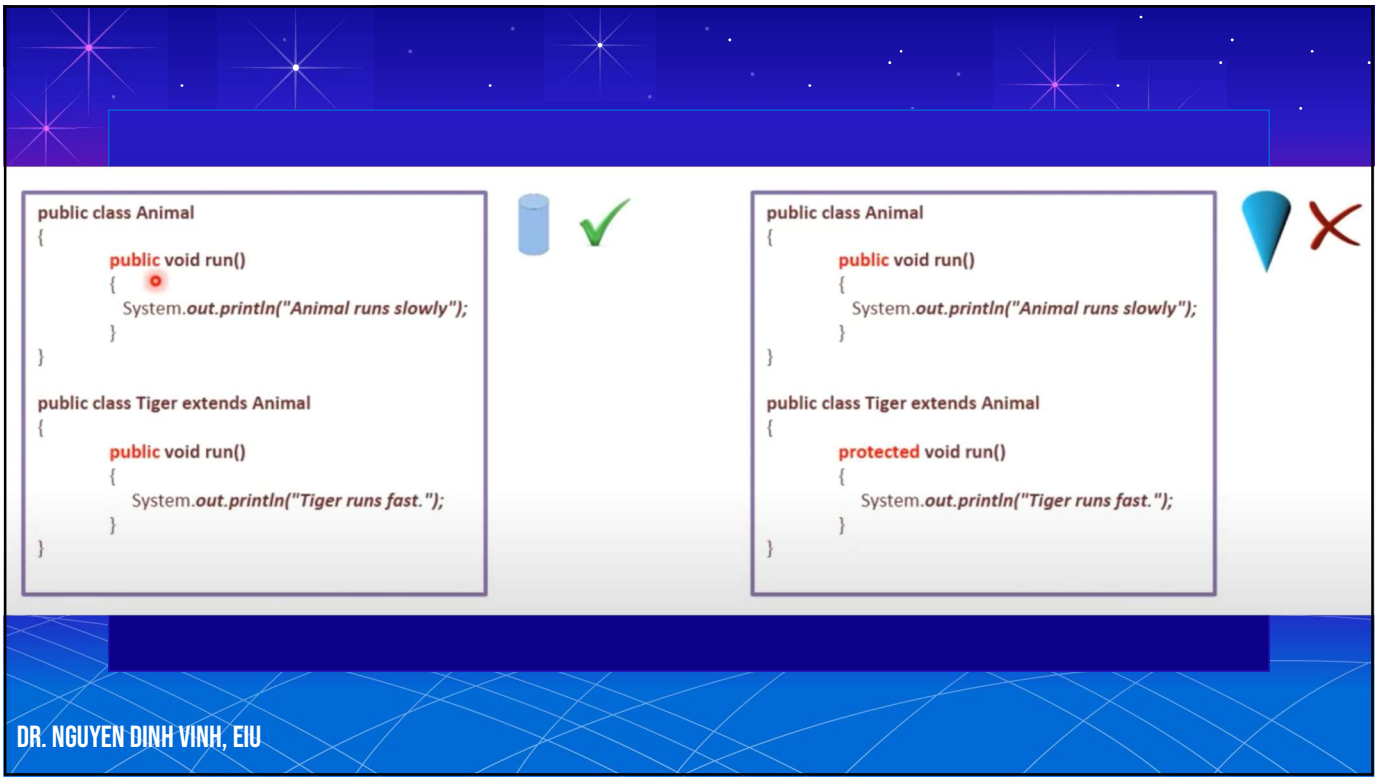
```
public class Animal
{
        protected void run()
        {
          System.out.println("Animal runs slowly");
        }
}

public class Tiger extends Animal
{
        public void run()
        {
           System.out.println("Tiger runs fast.");
        }
}
```

137

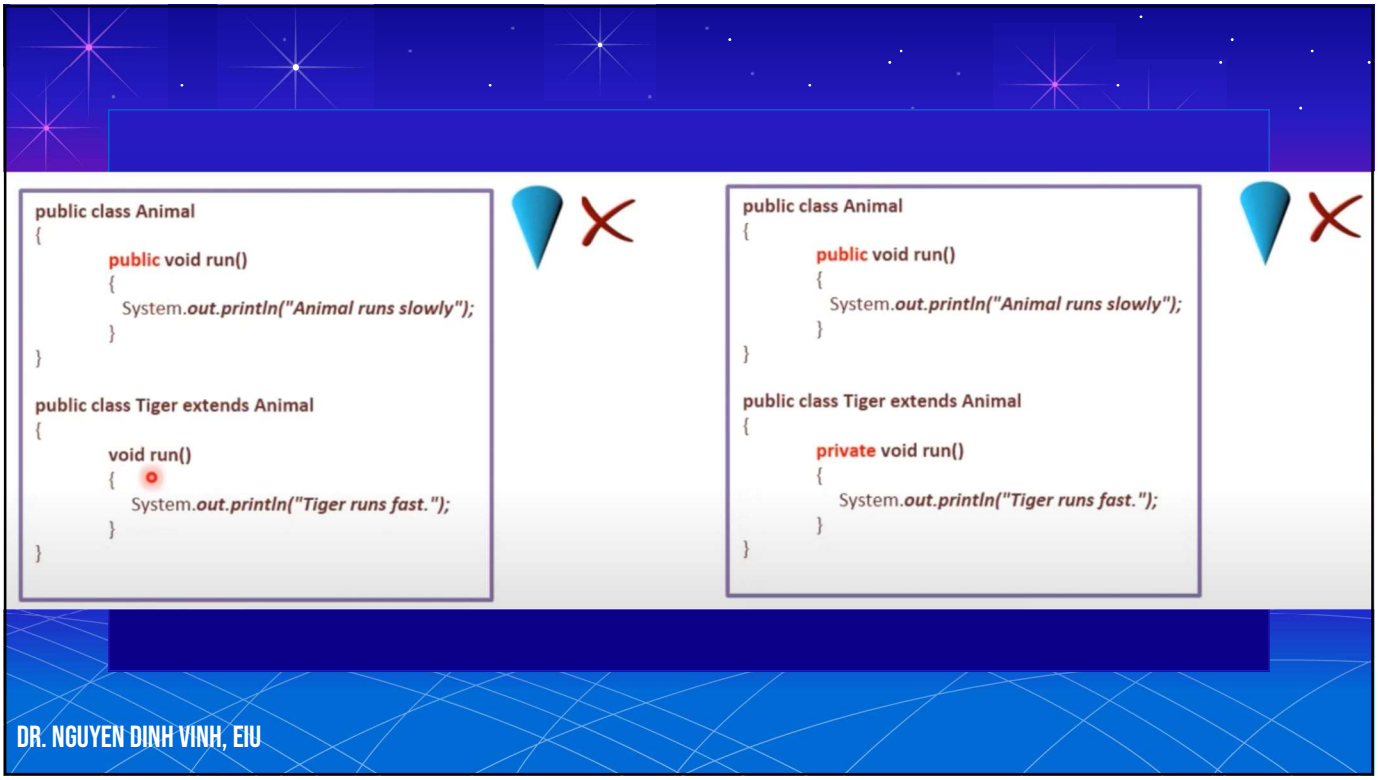
```
public class Animal
{
        protected void run()
        {
          System.out.println("Animal runs slowly");
        }
}

public class Tiger extends Animal
{
        void run()
        {
           System.out.println("Tiger runs fast.");
        }
}
```

```
public class Animal
{
        protected void run()
        {
          System.out.println("Animal runs slowly");
        }
}

public class Tiger extends Animal
{
        private void run()
        {
           System.out.println("Tiger runs fast.");
        }
}
```

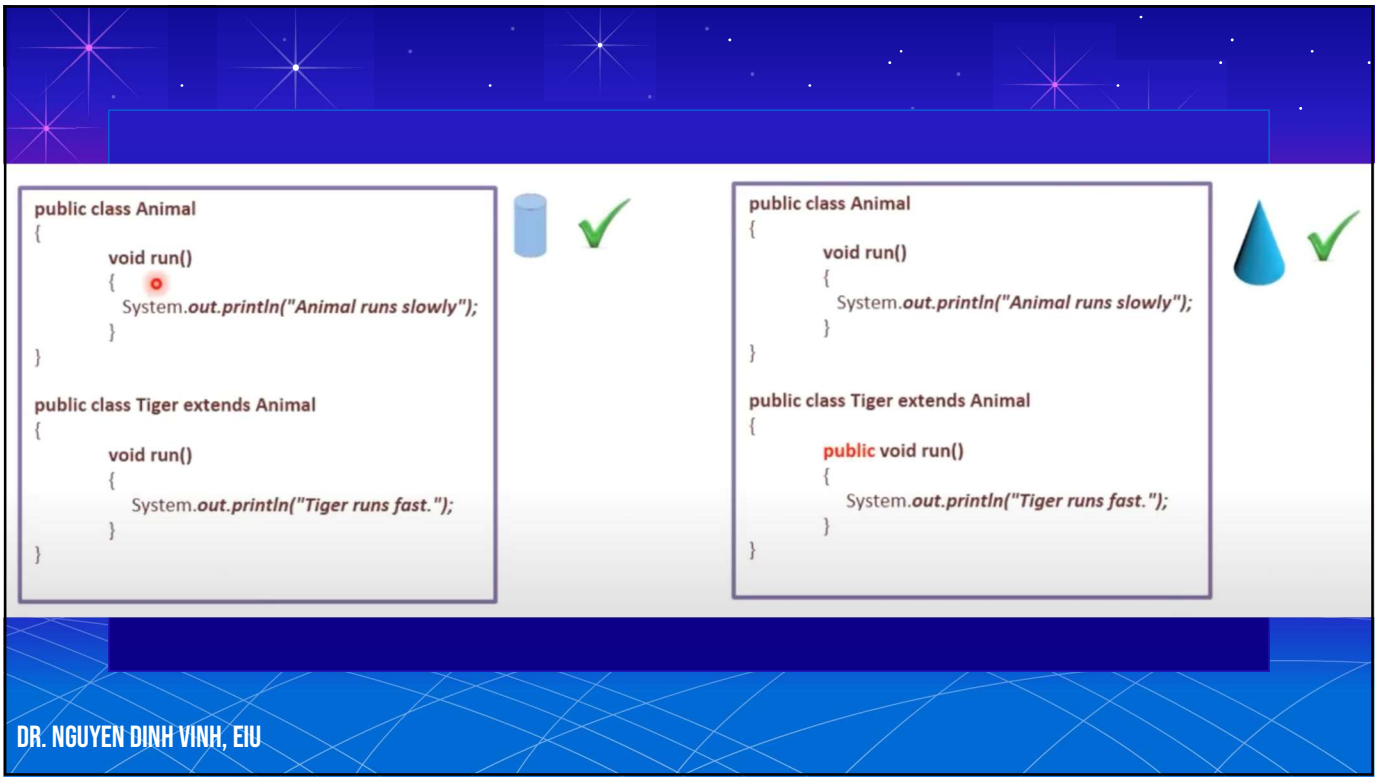
138


public class Animal
{
public void run()
{
System.out.println("Animal runs slowly");
}
}
public class Tiger extends Animal
{
public void run()
{
System.out.println("Tiger runs fast.");
}
}
public class Animal
{
public void run()
{
System.out.println("Animal runs slowly");
}
}
public class Tiger extends Animal
{
protected void run()
{
System.out.println("Tiger runs fast.");
}
}
DR. NGUYEN DINH VINH, EIU

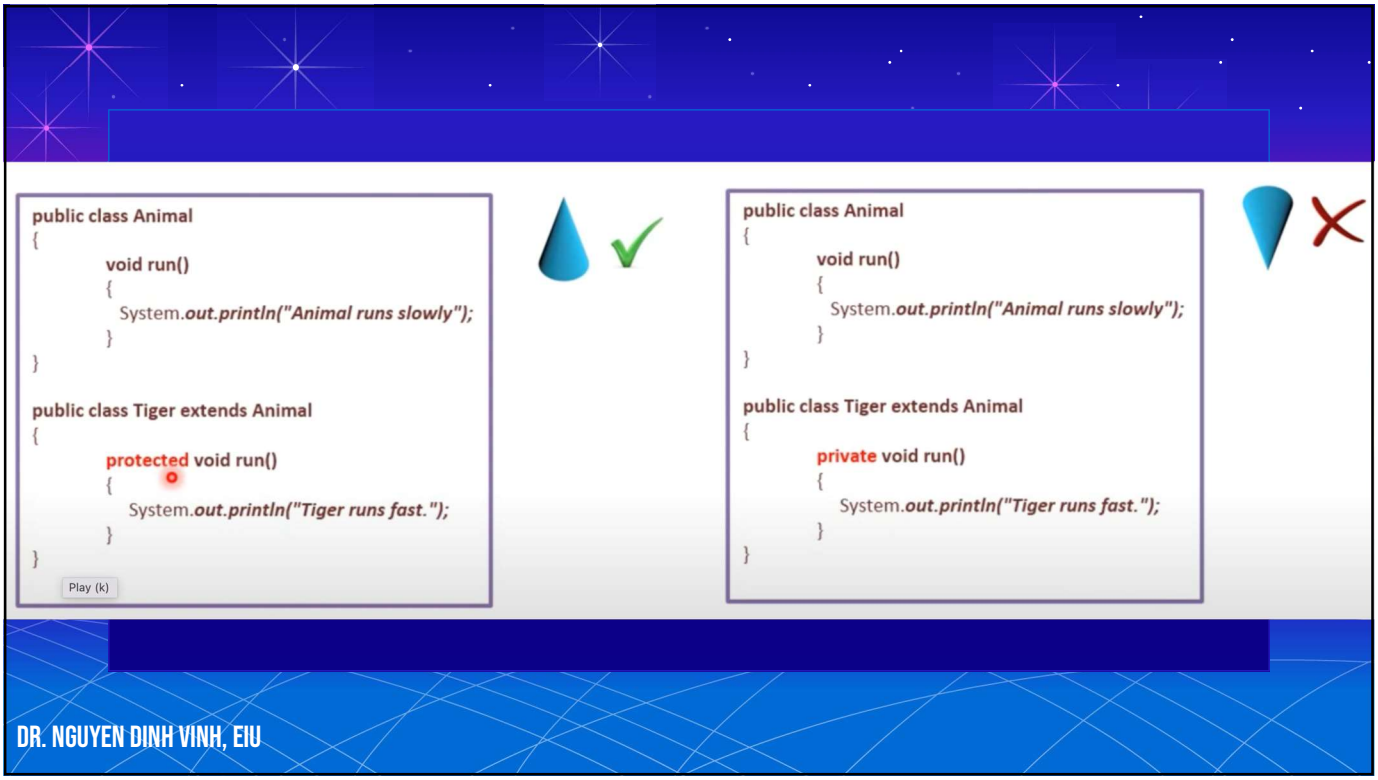
139


public class Animal
{
public void run()
{
System.out.println("Animal runs slowly");
}
}
public class Tiger extends Animal
{
void run()
{
System.out.println("Tiger runs fast.");
}
}
public class Animal
{
public void run()
{
System.out.println("Animal runs slowly");
}
}
public class Tiger extends Animal
{
private void run()
{
System.out.println("Tiger runs fast.");
}
}
DR. NGUYEN DINH VINH, EIU

140


public class Animal
{
void run()
{
System.out.println("Animal runs slowly");
}
}
public class Tiger extends Animal
{
void run()
{
System.out.println("Tiger runs fast.");
}
}
public class Animal
{
void run()
{
System.out.println("Animal runs slowly");
}
}
public class Tiger extends Animal
{
public void run()
{
System.out.println("Tiger runs fast.");
}
}
DR. NGUYEN DINH VINH, EIU

141


public class Animal
{
void run()
{
System.out.println("Animal runs slowly");
}
}
public class Tiger extends Animal
{
protected void run()
{
System.out.println("Tiger runs fast.");
}
}
Play (k)
public class Animal
{
void run()
{
System.out.println("Animal runs slowly");
}
}
public class Tiger extends Animal
{
private void run()
{
System.out.println("Tiger runs fast.");
}
}
DR. NGUYEN DINH VINH, EIU

142

Java method overriding (static modifier)

143

We will get a compile-time error if you attempt to change an instance method in the superclass to a static method in the subclass, and vice versa.

```
public class Animal
{
        public void run()
        {
           System.out.println("Animal runs slowly");
        }
}

public class Tiger extends Animal
{
        public static void run()
        {
          System.out.println("Tiger runs fast.");
        }
}
```

```
public class Animal
{
        public static void run()
        {
           System.out.println("Animal runs slowly");
        }
}

public class Tiger extends Animal
{
        public void run()
        {
          System.out.println("Tiger runs fast.");
        }
}
```

144

This is allowed because each static method belongs to its own class.

```
public class Animal
{
        public static void run()
        {
            System.out.println("Animal runs slowly");
        }
}

public class Tiger extends Animal
{
        public static void run()
        {
          System.out.println("Tiger runs fast.");
        }
}
```

145

Exercises

Build a polygon interface (IPolygon) including methods for calculating area and perimeter. Create classes Rectangle, circle, parallelogram, trapezoid... inherited from the IPolygon polygon interface.

146

References

- **E-Book**
  - Java Programming, 8th Edition Joyce Farrel
  - JDK Document 1.8
  - Java Tutorial
  - Java Tutorial Example
- **Website**
  - http://www.oracle.com/technetwork/articles/javase/jdk-netbeans-jsp-142931.html

Lecture notes: Msc. Nguyen Thanh Dai, EIU

DR. NGUYEN DINH VINH, EIU

147

THANK YOU

DR. NGUYEN DINH VINH, EIU

148