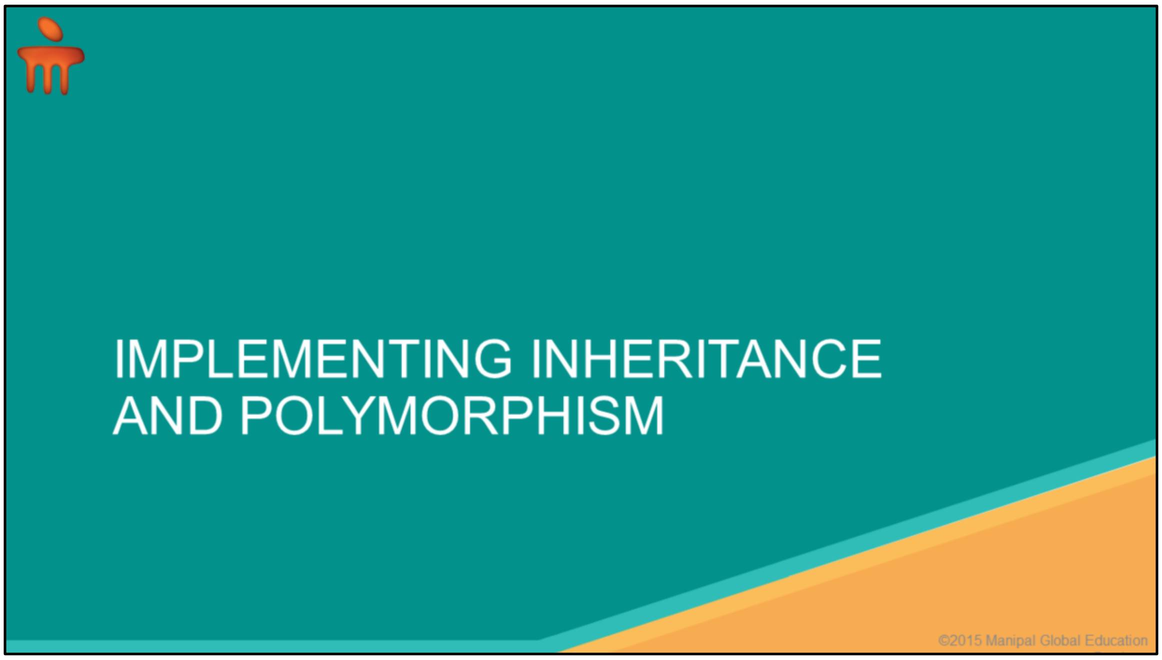
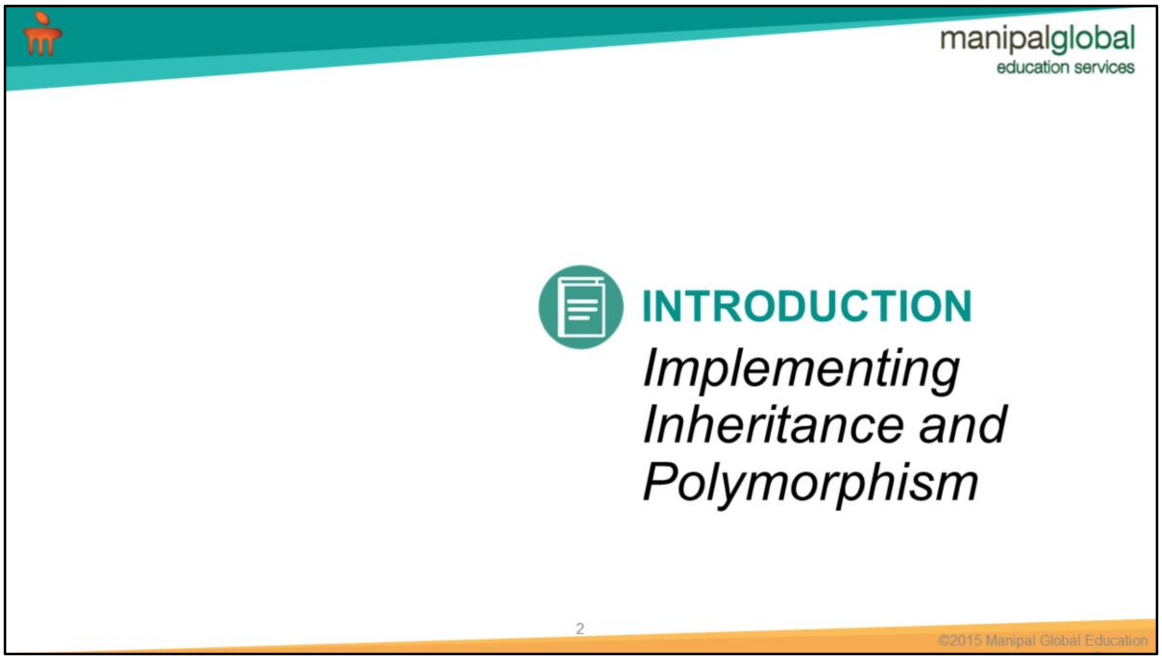


IMPLEMENTING INHERITANCE AND POLYMORPHISM

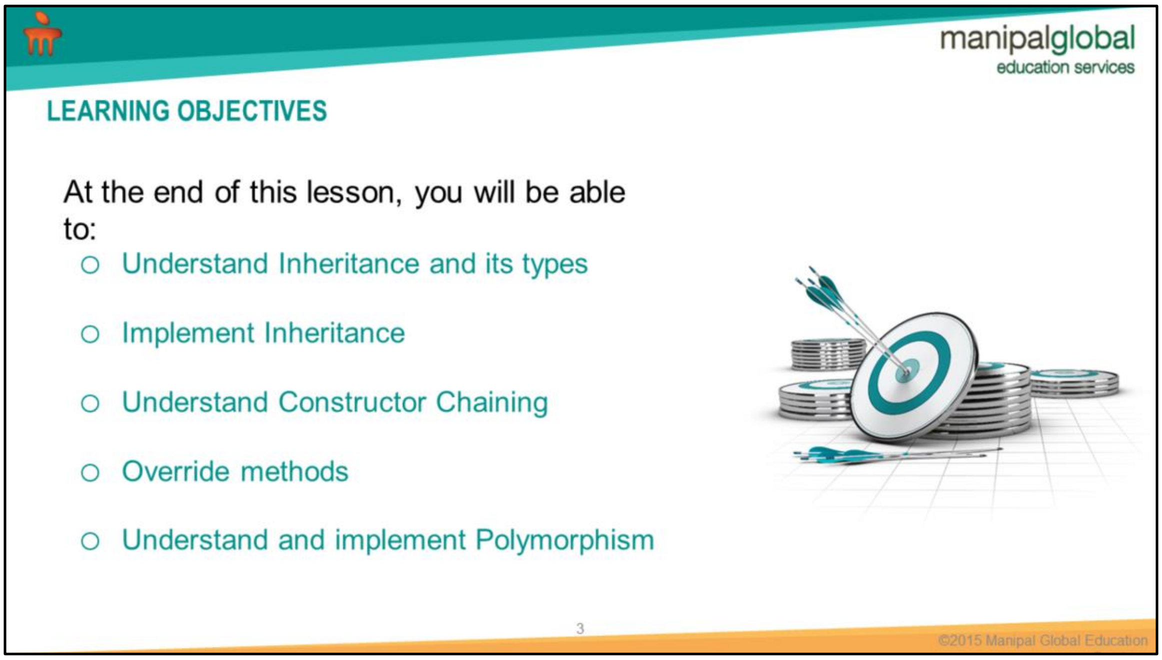


manipalglobal
education services

INTRODUCTION

Implementing Inheritance and Polymorphism

2



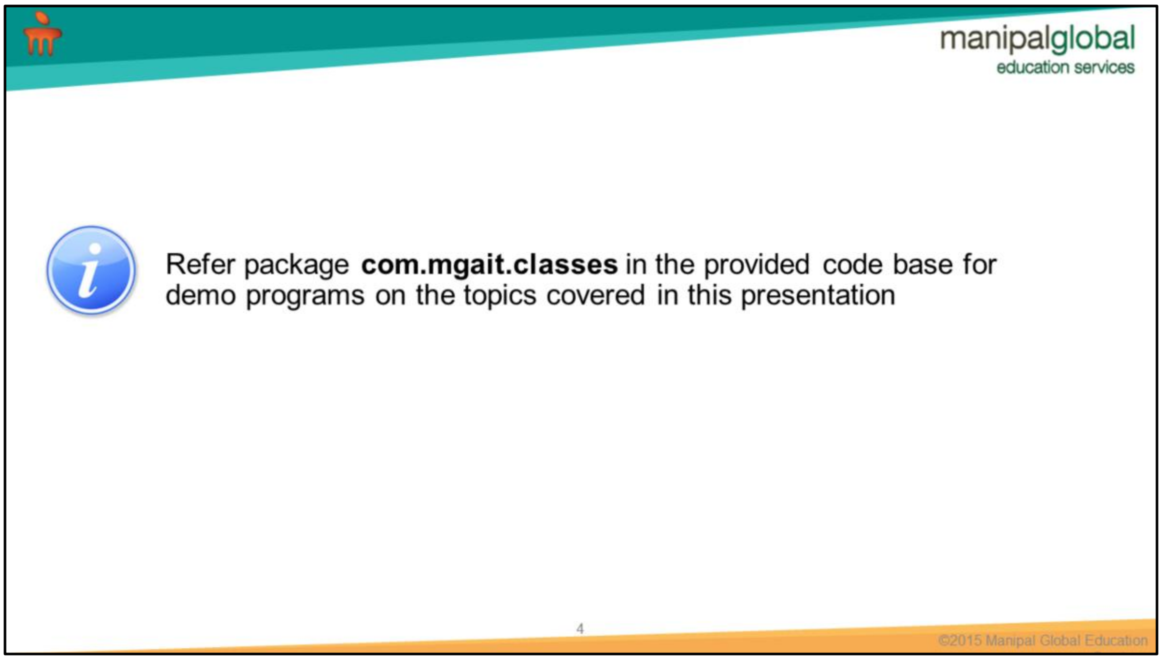
manipalglobal
education services

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

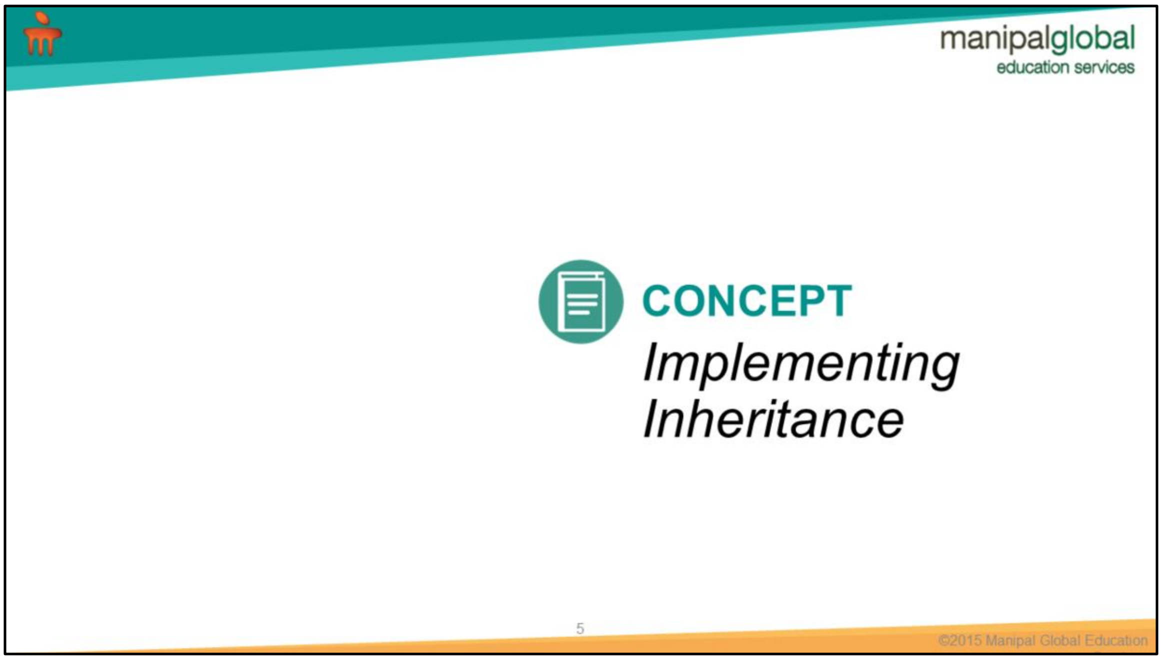
- Understand Inheritance and its types
- Implement Inheritance
- Understand Constructor Chaining
- Override methods
- Understand and implement Polymorphism

3



manipalglobal
education services

Refer package **com.mgait.classes** in the provided code base for demo programs on the topics covered in this presentation

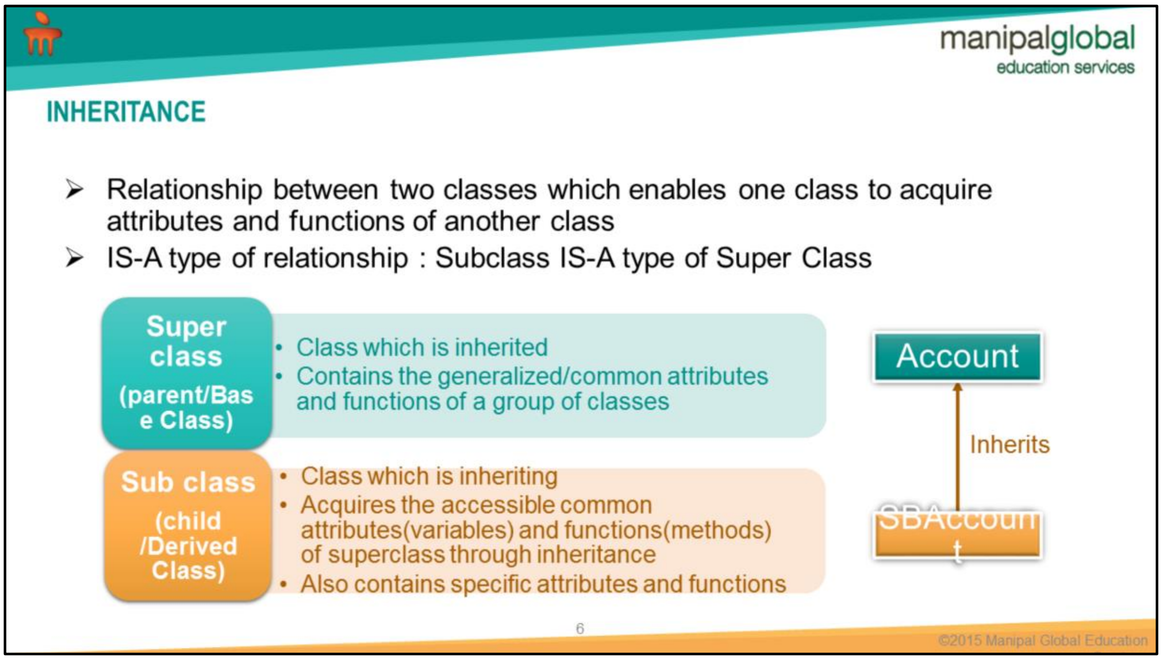


manipalglobal
education services

CONCEPT

Implementing Inheritance

5



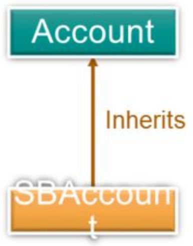
manipalglobal education services

INHERITANCE

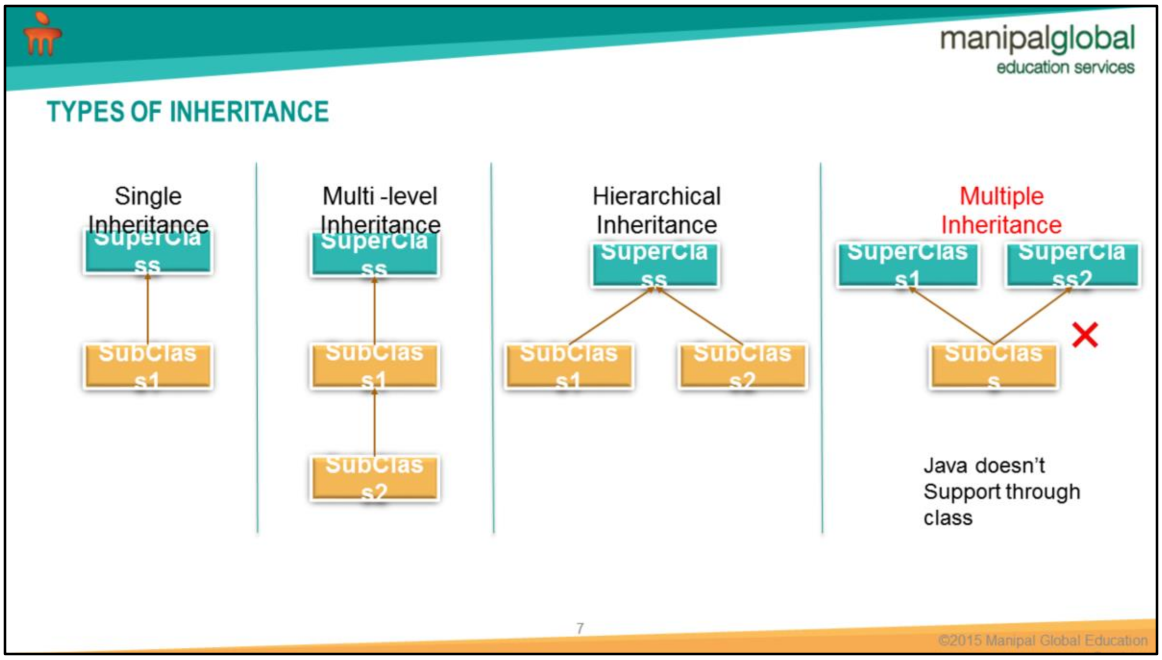
- Relationship between two classes which enables one class to acquire attributes and functions of another class
- IS-A type of relationship : Subclass IS-A type of Super Class

Super class (parent/Base Class)
- Class which is inherited
- Contains the generalized/common attributes and functions of a group of classes

Sub class (child /Derived Class)
- Class which is inheriting
- Acquires the accessible common attributes(variables) and functions(methods) of superclass through inheritance
- Also contains specific attributes and functions

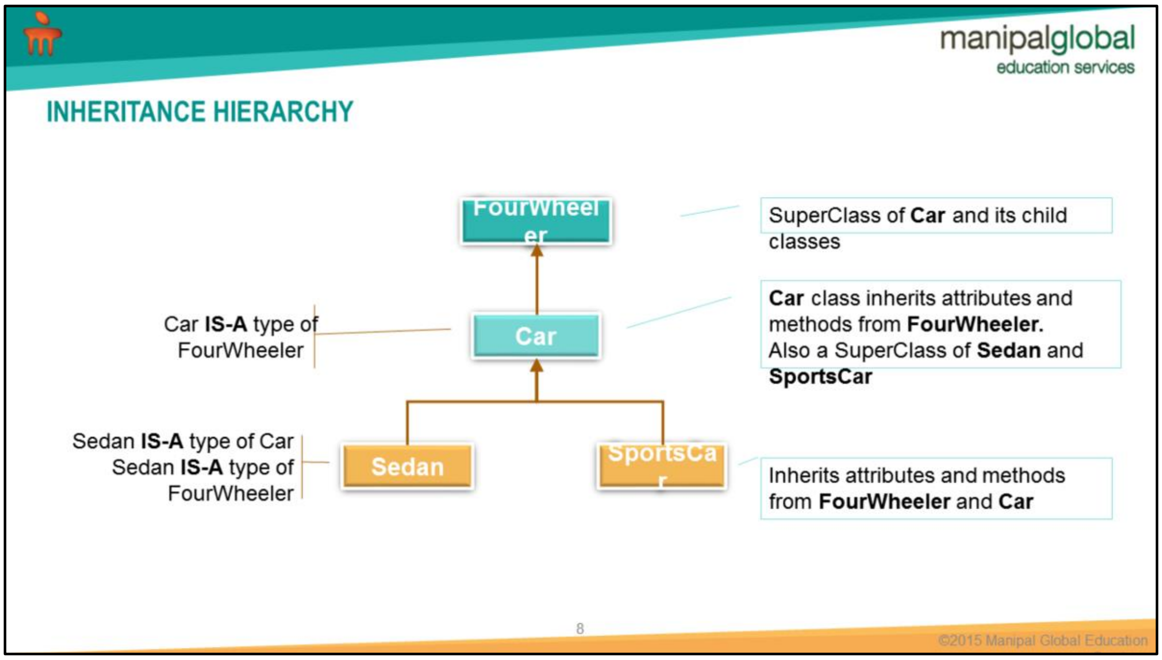


6



TYPES OF INHERITANCE

Java doesn't Support through class



INHERITANCE HIERARCHY

SuperClass of **Car** and its child classes

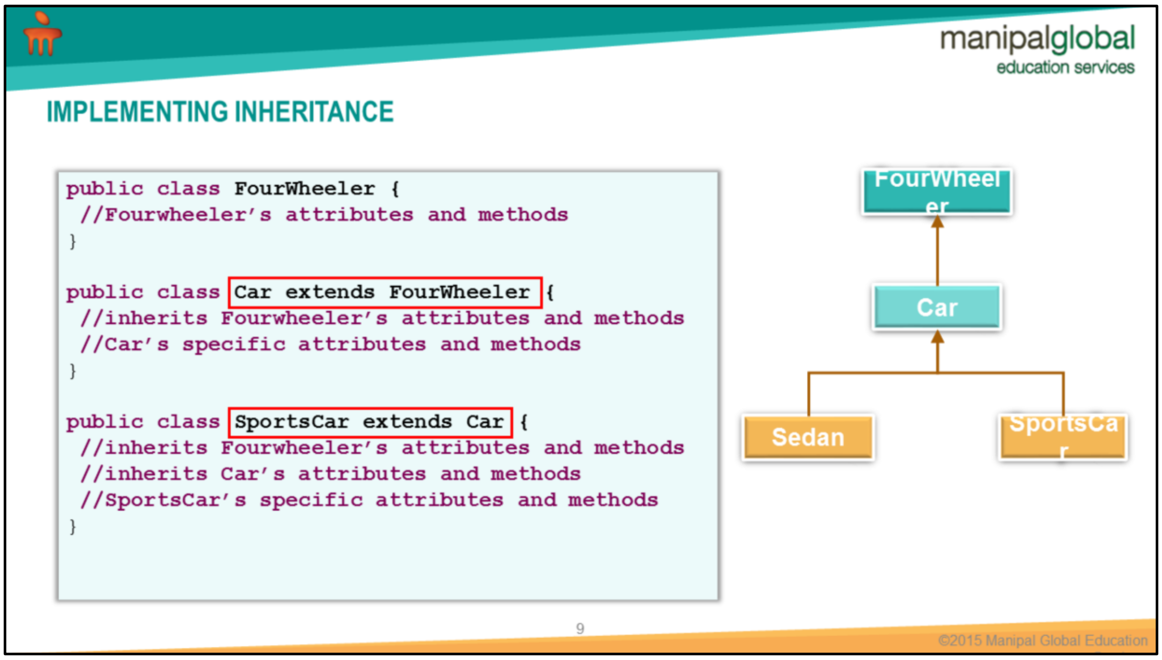
Car class inherits attributes and methods from **FourWheeler.**
Also a SuperClass of **Sedan** and **SportsCar**

Inherits attributes and methods from **FourWheeler** and **Car**

Car **IS-A** type of FourWheeler

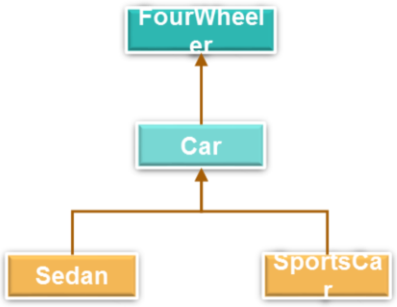
Sedan **IS-A** type of Car
Sedan **IS-A** type of FourWheeler

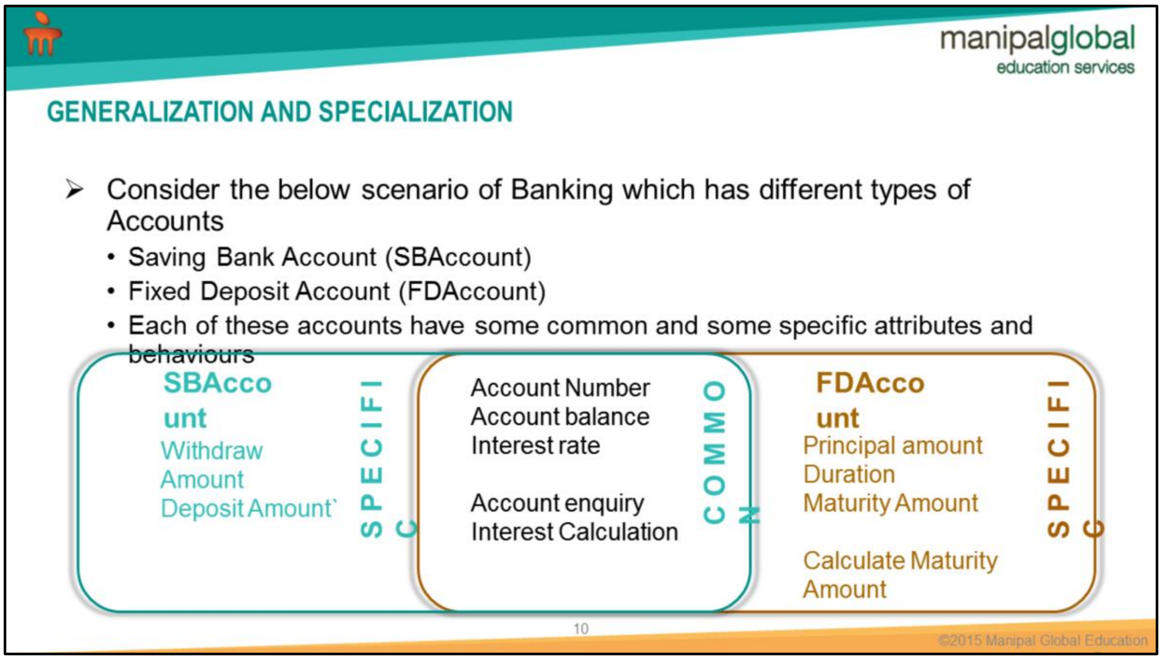
8



IMPLEMENTING INHERITANCE

```
public class FourWheeler {
 //Fourwheeler's attributes and methods
}

public class Car extends FourWheeler {
 //inherits Fourwheeler's attributes and methods
 //Car's specific attributes and methods
}

public class SportsCar extends Car {
 //inherits Fourwheeler's attributes and methods
 //inherits Car's attributes and methods
 //SportsCar's specific attributes and methods
}
```





manipalglobal
education services

GENERALIZATION AND SPECIALIZATION

- Consider the below scenario of Banking which has different types of Accounts
  - Saving Bank Account (SBAccount)
  - Fixed Deposit Account (FDAccount)
  - Each of these accounts have some common and some specific attributes and behaviours

SBAccount
Withdraw Amount
Deposit Amount`
SPECIFIC

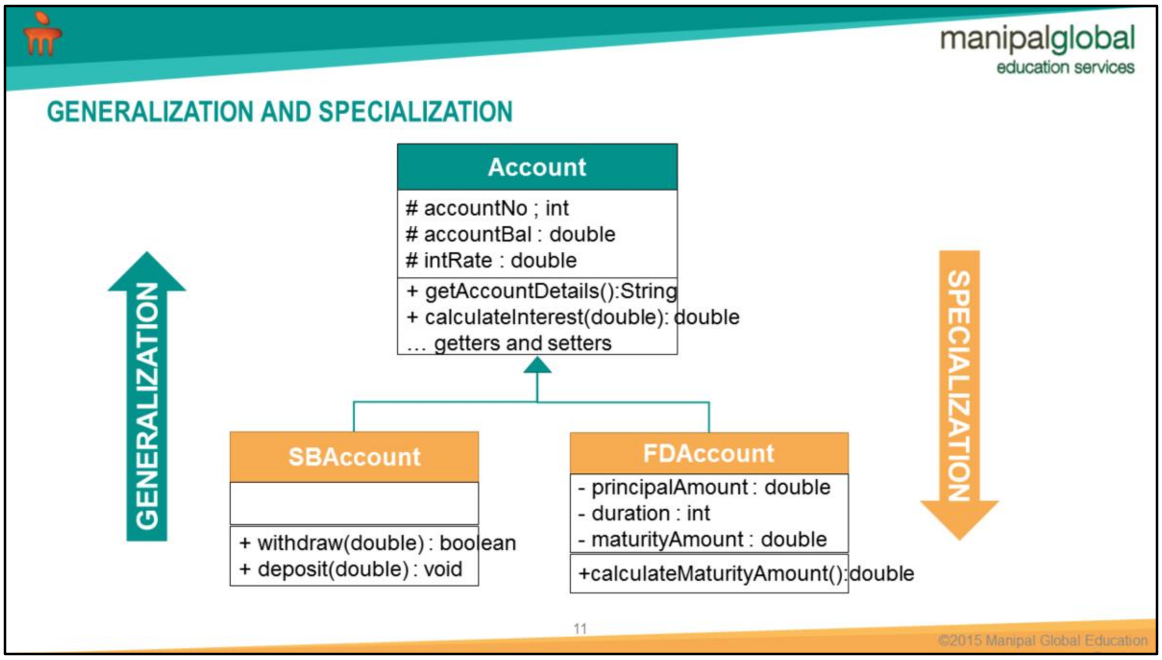
Account Number
Account balance
Interest rate

Account enquiry
Interest Calculation
COMMON

FDAccount
Principal amount
Duration
Maturity Amount

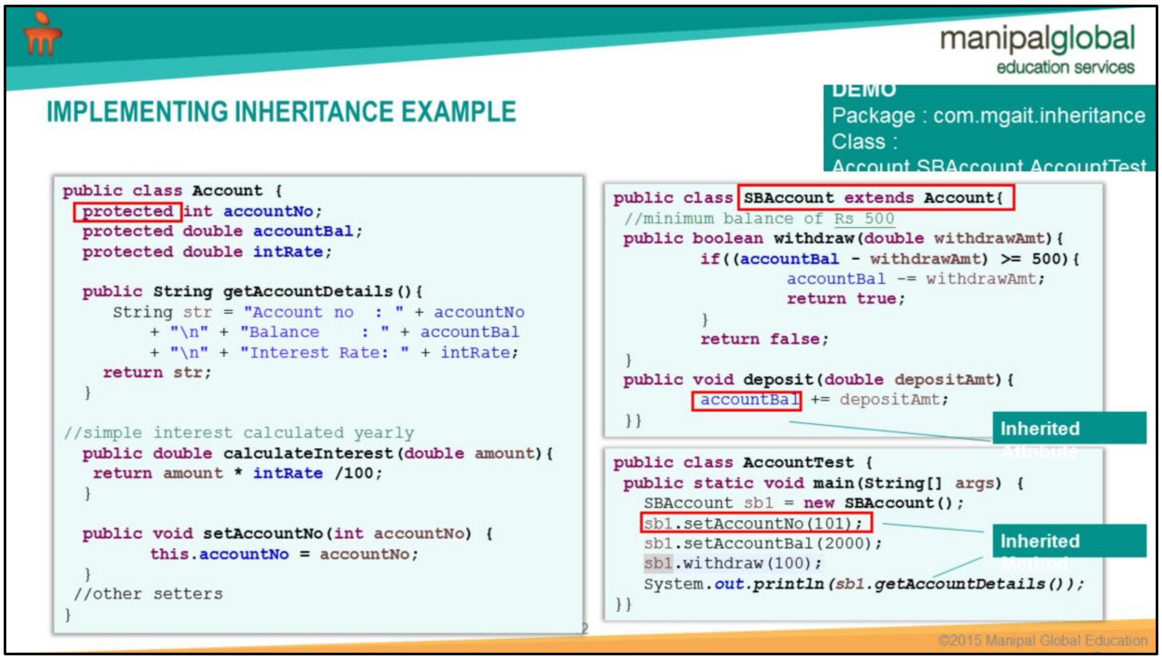
Calculate Maturity Amount
SPECIFIC

10



GENERALIZATION AND SPECIALIZATION

Account
# accountNo ; int
# accountBal : double
# intRate : double
+ getAccountDetails():String
+ calculateInterest(double): double
… getters and setters
SBAccount
+ withdraw(double) : boolean
+ deposit(double) : void
FDAccount
- principalAmount : double
- duration : int
- maturityAmount : double
+calculateMaturityAmount():double
GENERALIZATION
SPECIALIZATION


manipalglobal
education services

IMPLEMENTING INHERITANCE EXAMPLE

DEMO
Package : com.mgait.inheritance
Class :
Account,SBAccount,AccountTest

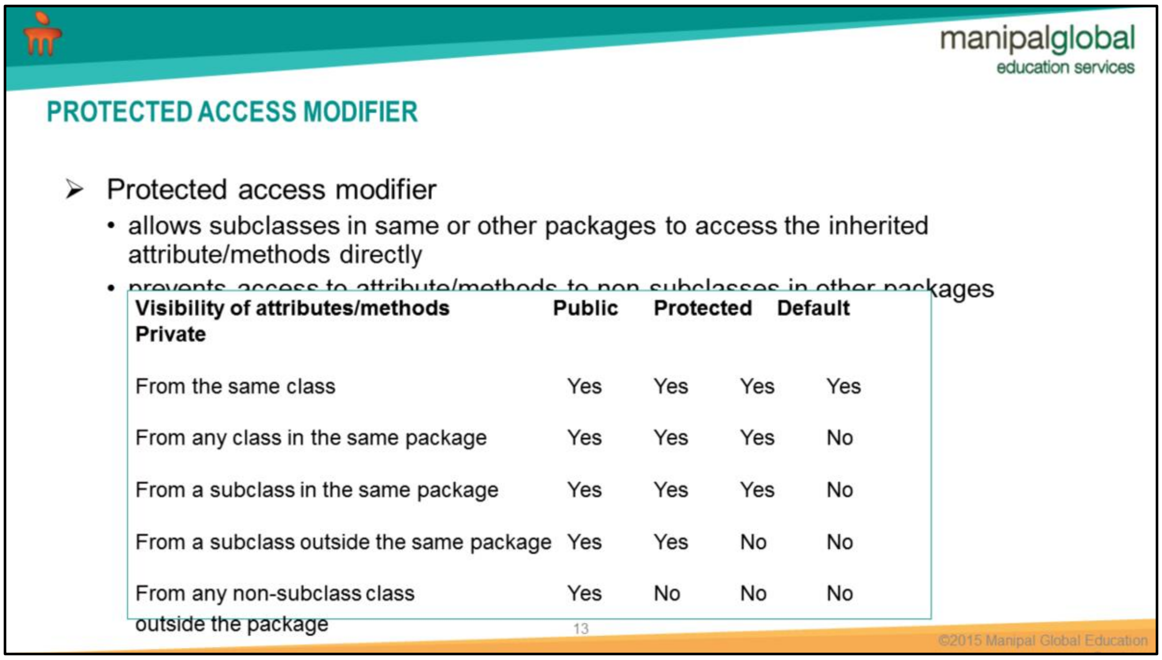
```
public class Account {
  protected int accountNo;
  protected double accountBal;
  protected double intRate;

  public String getAccountDetails(){
     String str = "Account no  : " + accountNo
         + "\n" + "Balance     : " + accountBal
         + "\n" + "Interest Rate: " + intRate;
    return str;
  }

//simple interest calculated yearly
  public double calculateInterest(double amount){
   return amount * intRate /100;
  }

  public void setAccountNo(int accountNo) {
         this.accountNo = accountNo;
  }
 //other setters
}
```

```
public class SBAccount extends Account{
 //minimum balance of Rs 500
 public boolean withdraw(double withdrawAmt){
        if((accountBal - withdrawAmt) >= 500){
                accountBal -= withdrawAmt;
                return true;
        }
        return false;
 }
 public void deposit(double depositAmt){
        accountBal += depositAmt;
 }}
```

Inherited Attribute

```
public class AccountTest {
 public static void main(String[] args) {
   SBAccount sb1 = new SBAccount();
   sb1.setAccountNo(101);
   sb1.setAccountBal(2000);
   sb1.withdraw(100);
   System.out.println(sb1.getAccountDetails());
}}
```

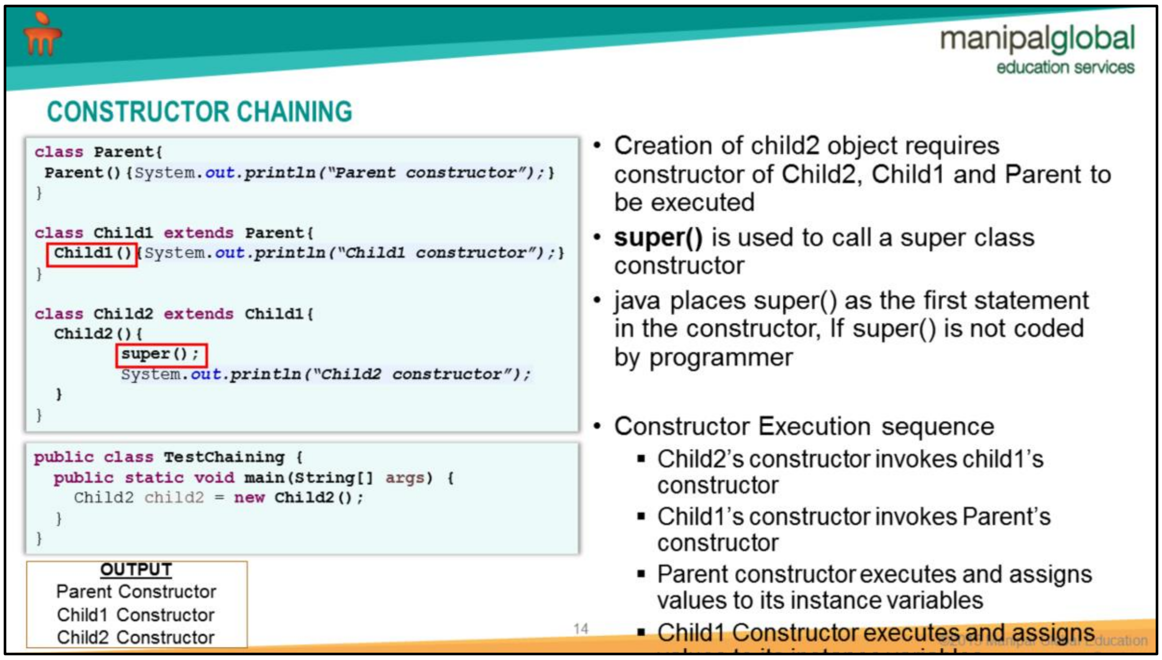
Inherited Method



PROTECTED ACCESS MODIFIER

- Protected access modifier
  - allows subclasses in same or other packages to access the inherited attribute/methods directly
  - prevents access to attribute/methods to non subclasses in other packages

Visibility of attributes/methods	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes	No	No
From any non-subclass class outside the package	Yes	No	No	No


manipalglobal
education services

CONSTRUCTOR CHAINING

```
class Parent{
 Parent(){System.out.println("Parent constructor");}
}

class Child1 extends Parent{
  Child1(){System.out.println("Child1 constructor");}
}

class Child2 extends Child1{
  Child2(){
         super();
         System.out.println("Child2 constructor");
  }
}
```

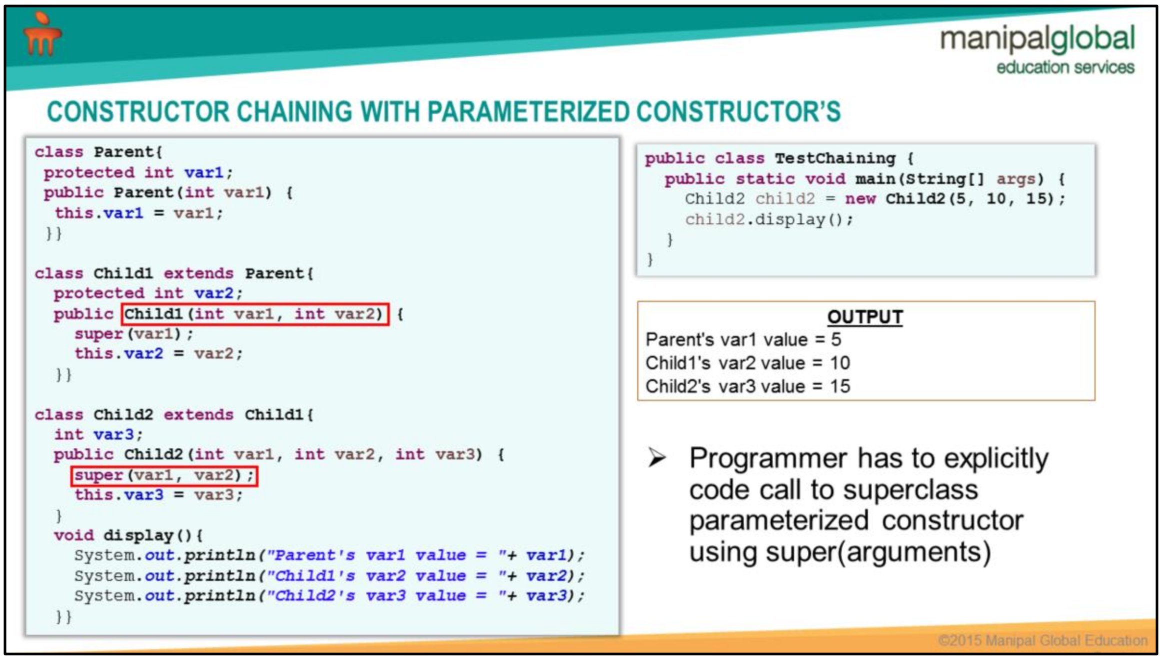
```
public class TestChaining {
  public static void main(String[] args) {
    Child2 child2 = new Child2();
  }
}
```

OUTPUT

```
Parent Constructor
Child1 Constructor
Child2 Constructor
```

- Creation of child2 object requires constructor of Child2, Child1 and Parent to be executed
- **super()** is used to call a super class constructor
- java places super() as the first statement in the constructor, If super() is not coded by programmer

- Constructor Execution sequence
  - Child2's constructor invokes child1's constructor
  - Child1's constructor invokes Parent's constructor
  - Parent constructor executes and assigns values to its instance variables
  - Child1 Constructor executes and assigns



manipalglobal
education services

CONSTRUCTOR CHAINING WITH PARAMETERIZED CONSTRUCTOR'S

```
class Parent{
 protected int var1;
 public Parent(int var1) {
  this.var1 = var1;
 }}

class Child1 extends Parent{
  protected int var2;
  public Child1(int var1, int var2) {
    super(var1);
    this.var2 = var2;
  }}

class Child2 extends Child1{
  int var3;
  public Child2(int var1, int var2, int var3) {
    super(var1, var2);
    this.var3 = var3;
  }
  void display(){
    System.out.println("Parent's var1 value = "+ var1);
    System.out.println("Child1's var2 value = "+ var2);
    System.out.println("Child2's var3 value = "+ var3);
  }}
```

```
public class TestChaining {
  public static void main(String[] args) {
    Child2 child2 = new Child2(5, 10, 15);
    child2.display();
  }
}
```

OUTPUT

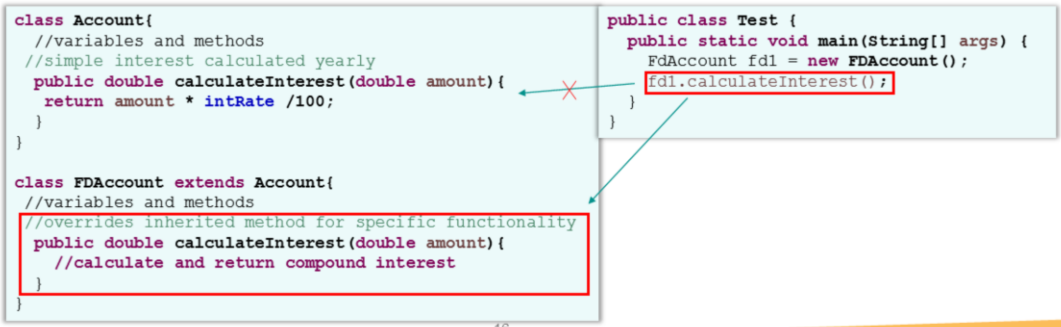
```
Parent's var1 value = 5
Child1's var2 value = 10
Child2's var3 value = 15
```

- Programmer has to explicitly code call to superclass parameterized constructor using super(arguments)

manipalglobal
education services

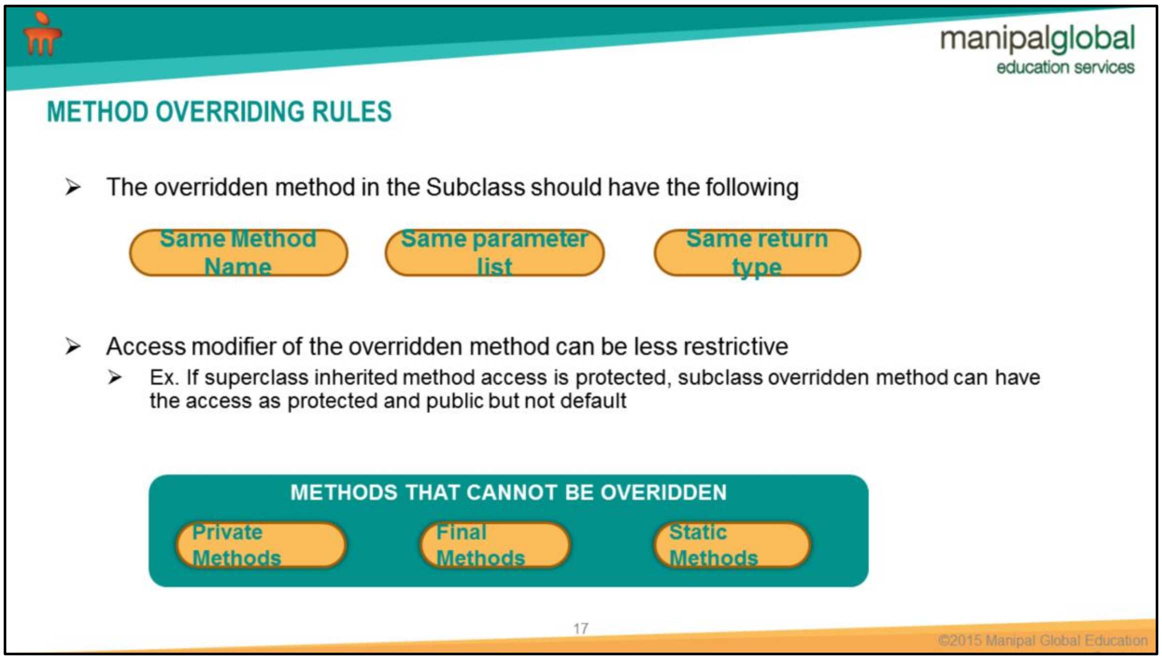
METHOD OVERRIDING

- Subclass can override inherited methods of Superclass
- Why
  - To define behavior that's specific to a particular subclass



```
class Account{
  //variables and methods
 //simple interest calculated yearly
  public double calculateInterest(double amount){
   return amount * intRate /100;
  }
}

class FDAccount extends Account{
 //variables and methods
 //overrides inherited method for specific functionality
  public double calculateInterest(double amount){
    //calculate and return compound interest
  }
}
```

```
public class Test {
  public static void main(String[] args) {
    FdAccount fd1 = new FDAccount();
    fd1.calculateInterest();
  }
}
```



manipalglobal
education services

METHOD OVERRIDING RULES

- The overridden method in the Subclass should have the following

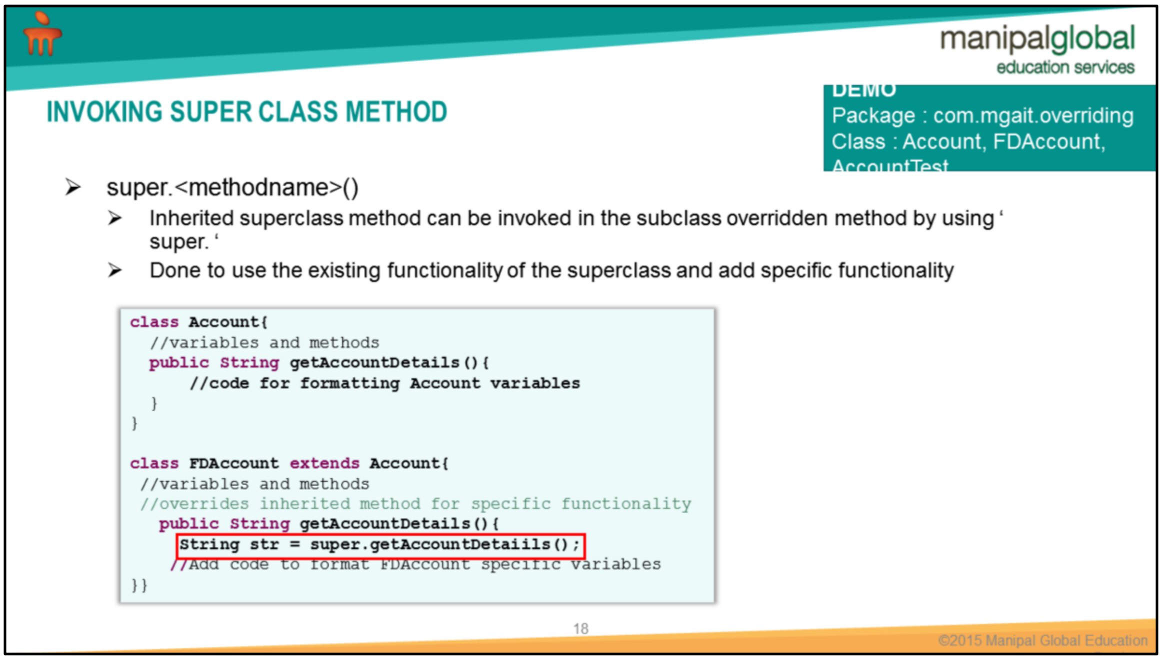
Same Method Name | Same parameter list | Same return type

- Access modifier of the overridden method can be less restrictive
  - Ex. If superclass inherited method access is protected, subclass overridden method can have the access as protected and public but not default

METHODS THAT CANNOT BE OVERIDDEN

Private Methods | Final Methods | Static Methods

17



manipalglobal
education services

INVOKING SUPER CLASS METHOD

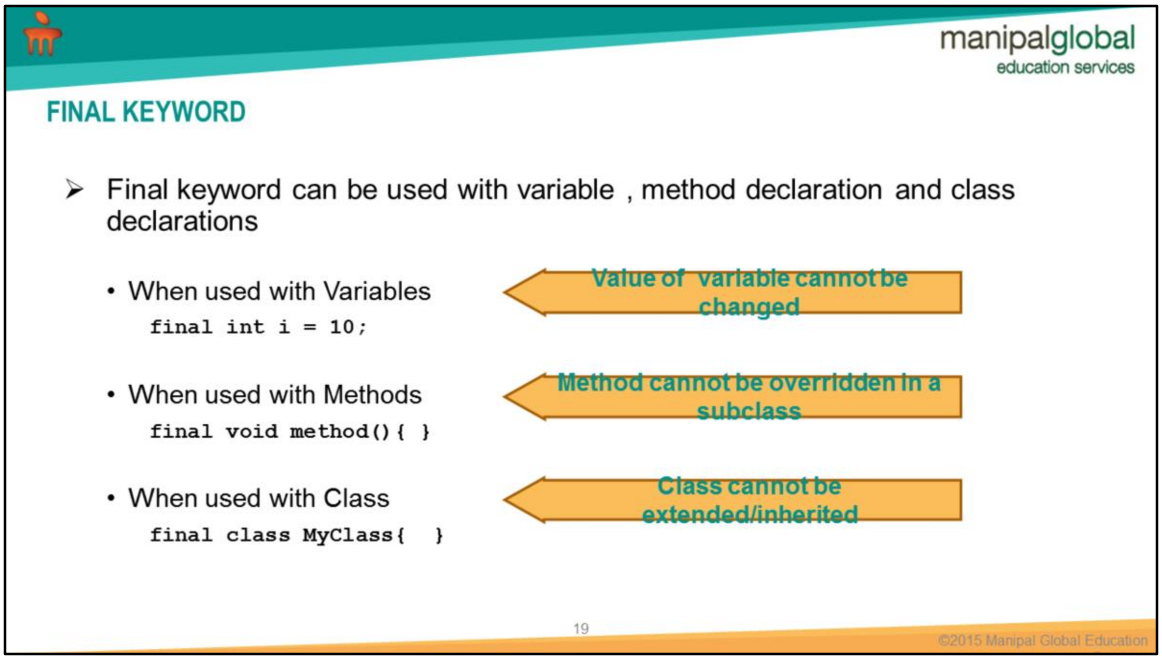
DEMO
Package : com.mgait.overriding
Class : Account, FDAccount, AccountTest

- super.<methodname>()
  - Inherited superclass method can be invoked in the subclass overridden method by using ' super. '
  - Done to use the existing functionality of the superclass and add specific functionality

```
class Account{
  //variables and methods
  public String getAccountDetails(){
      //code for formatting Account variables
  }
}

class FDAccount extends Account{
 //variables and methods
 //overrides inherited method for specific functionality
   public String getAccountDetails(){
     String str = super.getAccountDetaiils();
     //Add code to format FDAccount specific variables
}}
```

18

©2015 Manipal Global Education



manipalglobal
education services

FINAL KEYWORD

➢ Final keyword can be used with variable , method declaration and class declarations

- When used with Variables

```
final int i = 10;
```

Value of variable cannot be changed

- When used with Methods

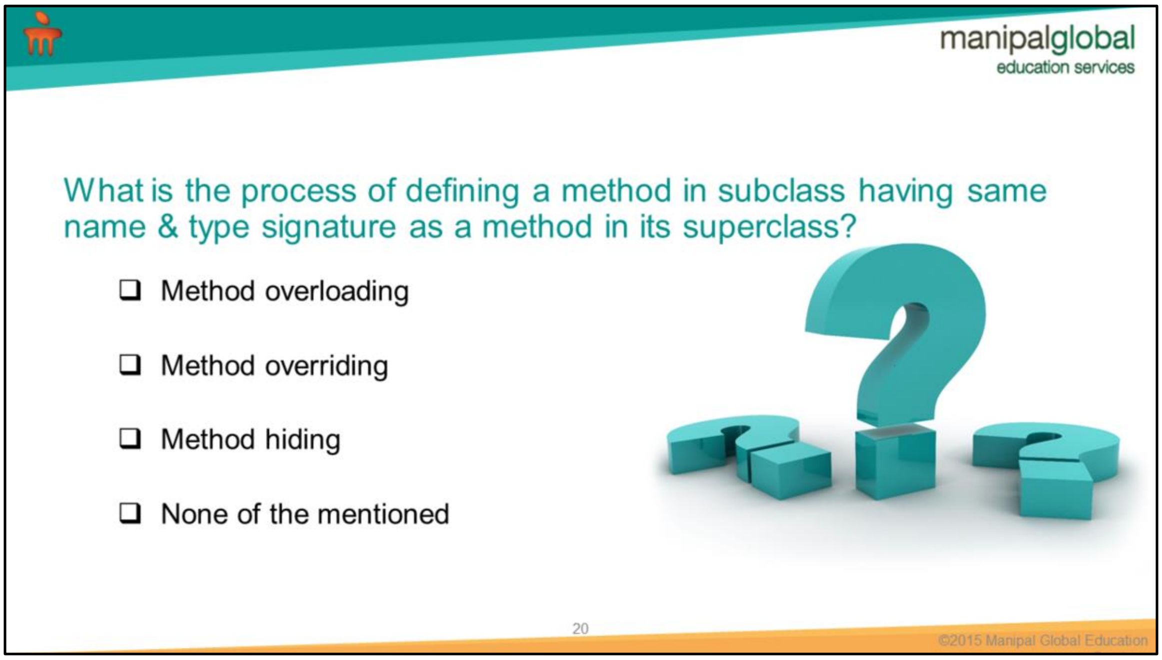
```
final void method(){ }
```

Method cannot be overridden in a subclass

- When used with Class

```
final class MyClass{   }
```

Class cannot be extended/inherited



manipalglobal
education services

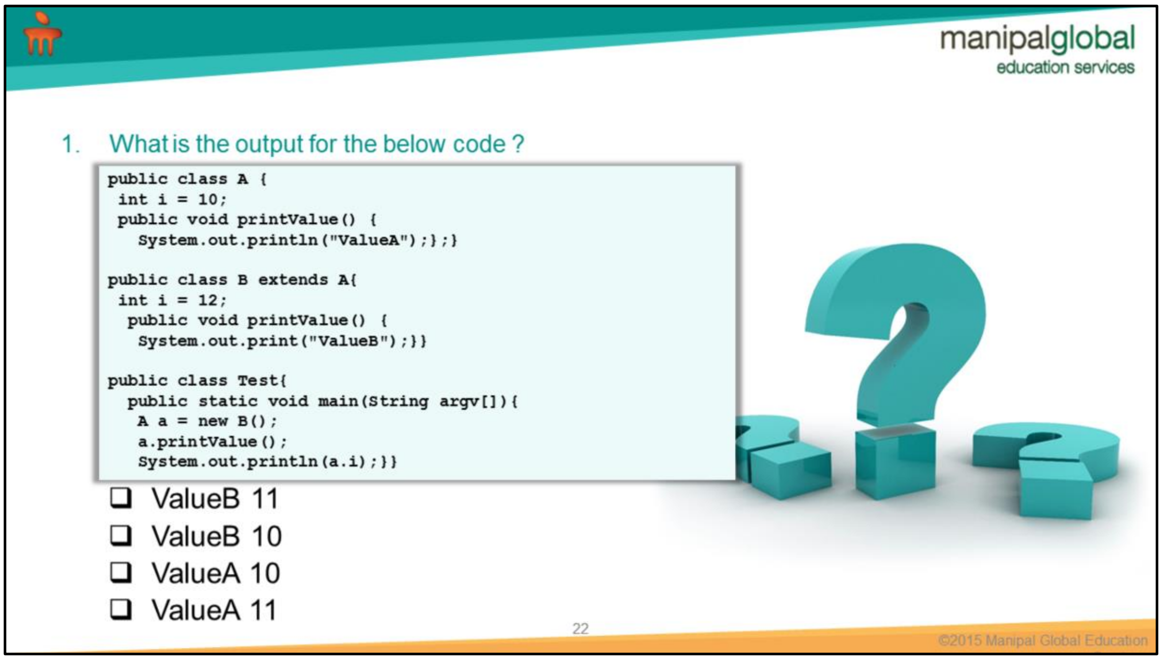
What is the process of defining a method in subclass having same name & type signature as a method in its superclass?

- ❑ Method overloading
- ❑ Method overriding
- ❑ Method hiding
- ❑ None of the mentioned

manipalglobal
education services

Object oriented inheritance models the

- ❑ "is a " relationship
- ❑ "has a" relationship
- ❑ inheritance does not describe any kind of relationship between classes
- ❑ All of the above



manipalglobal
education services

1. What is the output for the below code ?

```
public class A {
 int i = 10;
 public void printValue() {
   System.out.println("ValueA");};};

public class B extends A{
 int i = 12;
  public void printValue()  {
   System.out.print("ValueB");}}

public class Test{
  public static void main(String argv[]){
   A a = new B();
   a.printValue();
   System.out.println(a.i);}}
```

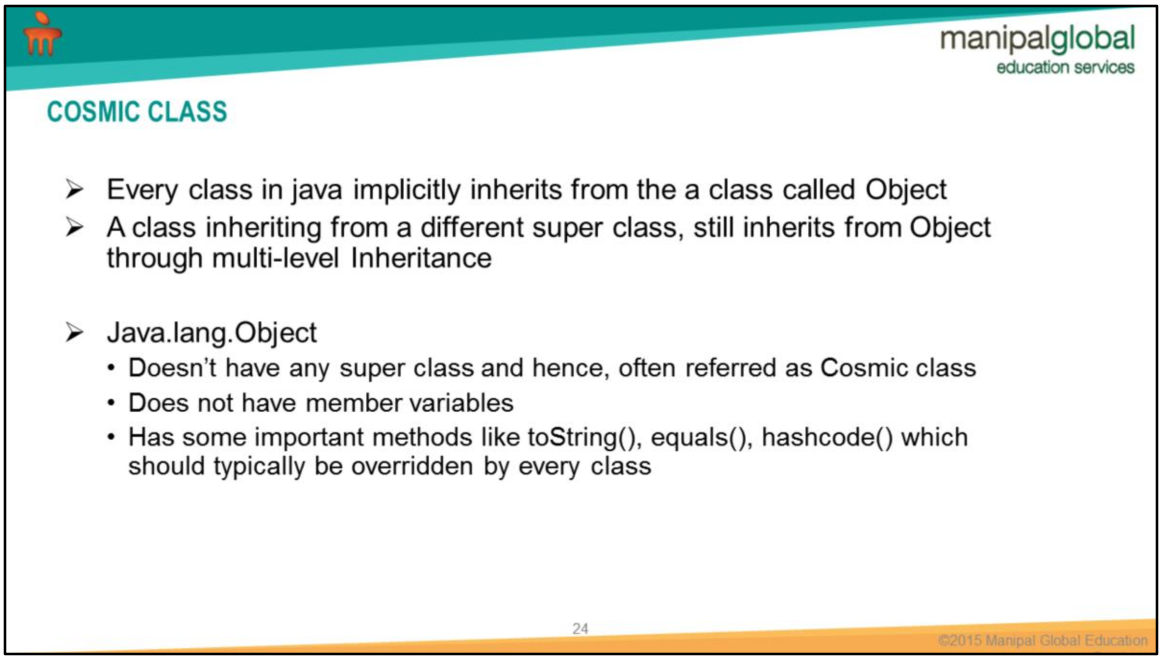
- ❑ ValueB 11
- ❑ ValueB 10
- ❑ ValueA 10
- ❑ ValueA 11

22

manipalglobal
education services

CONCEPT

Cosmic Class

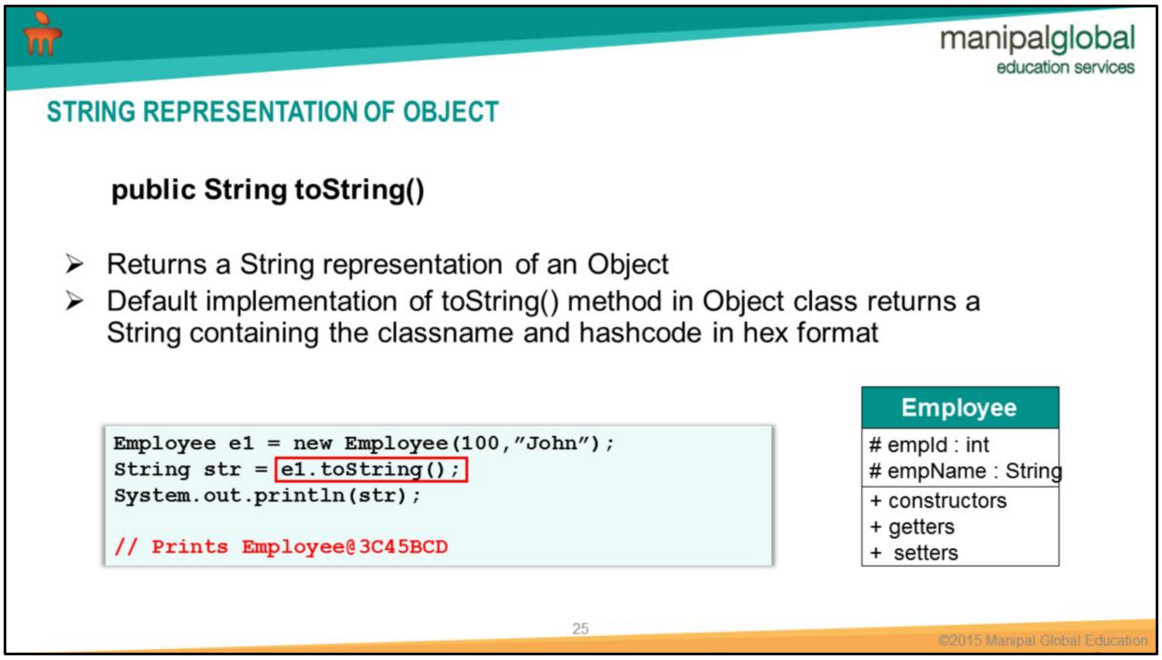


manipalglobal
education services

COSMIC CLASS

- ➢ Every class in java implicitly inherits from the a class called Object
- ➢ A class inheriting from a different super class, still inherits from Object through multi-level Inheritance

- ➢ Java.lang.Object
  - Doesn't have any super class and hence, often referred as Cosmic class
  - Does not have member variables
  - Has some important methods like toString(), equals(), hashcode() which should typically be overridden by every class



manipalglobal
education services

STRING REPRESENTATION OF OBJECT

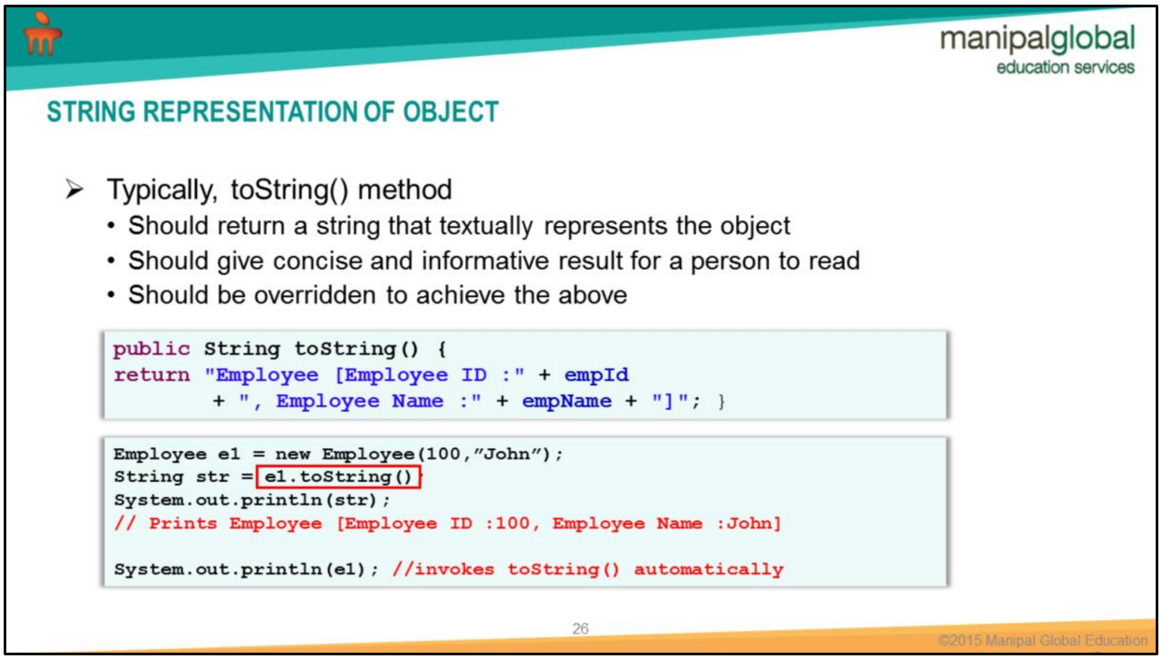
public String toString()

- Returns a String representation of an Object
- Default implementation of toString() method in Object class returns a String containing the classname and hashcode in hex format

```
Employee e1 = new Employee(100,"John");
String str = e1.toString();
System.out.println(str);

// Prints Employee@3C45BCD
```

Employee
empId : int # empName : String
+ constructors + getters + setters

25

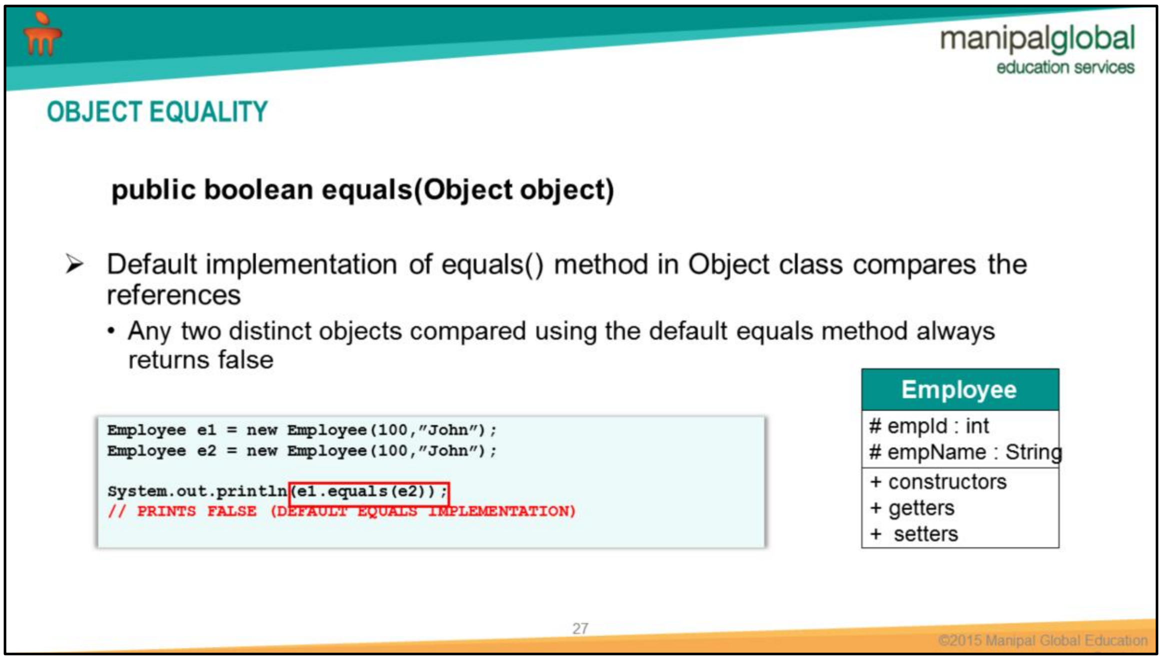


STRING REPRESENTATION OF OBJECT

- Typically, toString() method
  - Should return a string that textually represents the object
  - Should give concise and informative result for a person to read
  - Should be overridden to achieve the above

```
public String toString() {
return "Employee [Employee ID :" + empId
        + ", Employee Name :" + empName + "]"; }
```

```
Employee e1 = new Employee(100,"John");
String str = e1.toString();
System.out.println(str);
// Prints Employee [Employee ID :100, Employee Name :John]

System.out.println(e1); //invokes toString() automatically
```



manipalglobal
education services

OBJECT EQUALITY

public boolean equals(Object object)

- Default implementation of equals() method in Object class compares the references
  - Any two distinct objects compared using the default equals method always returns false

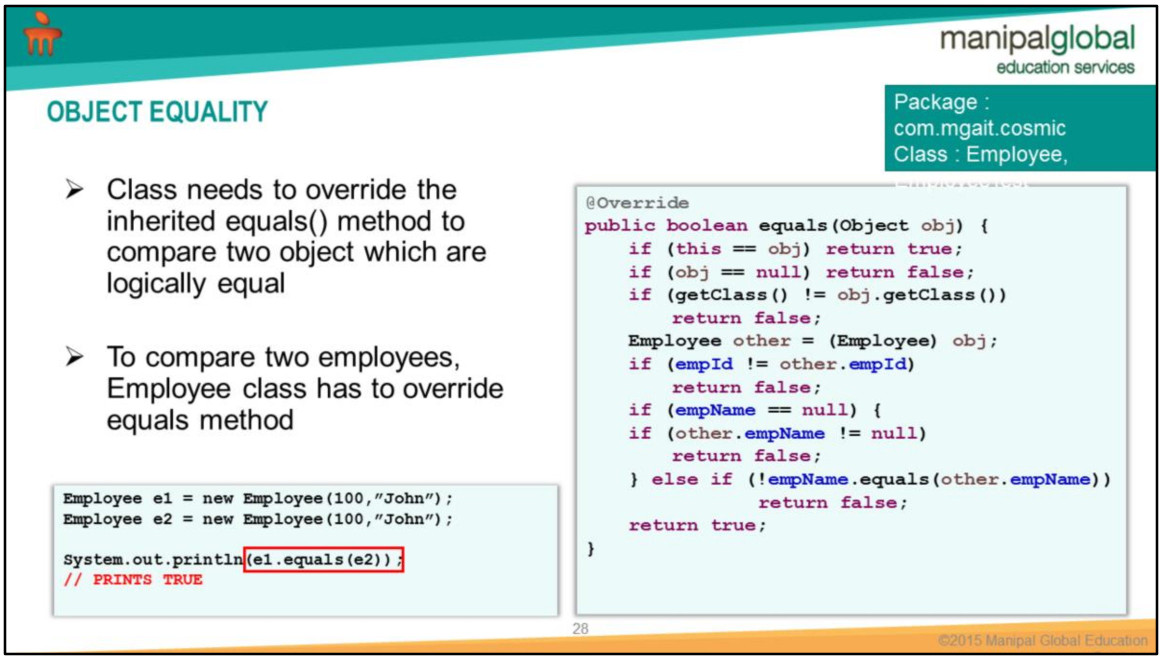
```
Employee e1 = new Employee(100,"John");
Employee e2 = new Employee(100,"John");

System.out.println(e1.equals(e2));
// PRINTS FALSE (DEFAULT EQUALS IMPLEMENTATION)
```

Employee

empId : int
empName : String

+ constructors
+ getters
+ setters

27



manipalglobal
education services

OBJECT EQUALITY

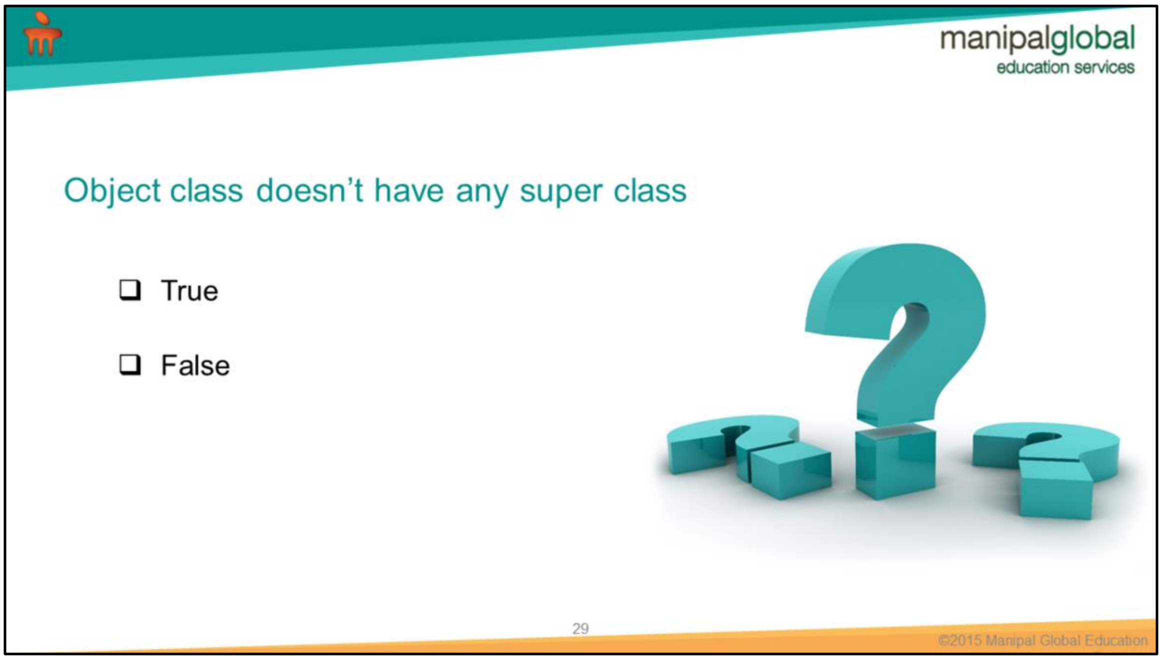
Package :
com.mgait.cosmic
Class : Employee,

- Class needs to override the inherited equals() method to compare two object which are logically equal
- To compare two employees, Employee class has to override equals method

```
Employee e1 = new Employee(100,"John");
Employee e2 = new Employee(100,"John");

System.out.println(e1.equals(e2));
// PRINTS TRUE
```

```
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null) return false;
    if (getClass() != obj.getClass())
        return false;
    Employee other = (Employee) obj;
    if (empId != other.empId)
        return false;
    if (empName == null) {
    if (other.empName != null)
        return false;
    } else if (!empName.equals(other.empName))
            return false;
    return true;
}
```

28

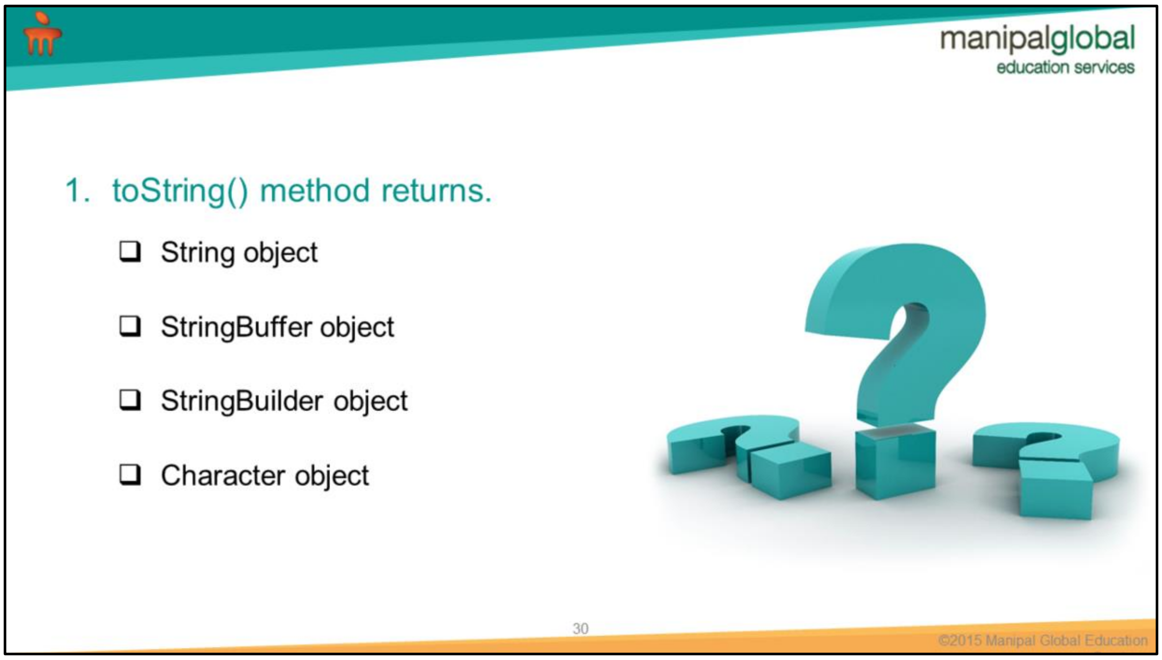


manipalglobal
education services

Object class doesn't have any super class

- ❑ True
- ❑ False

29

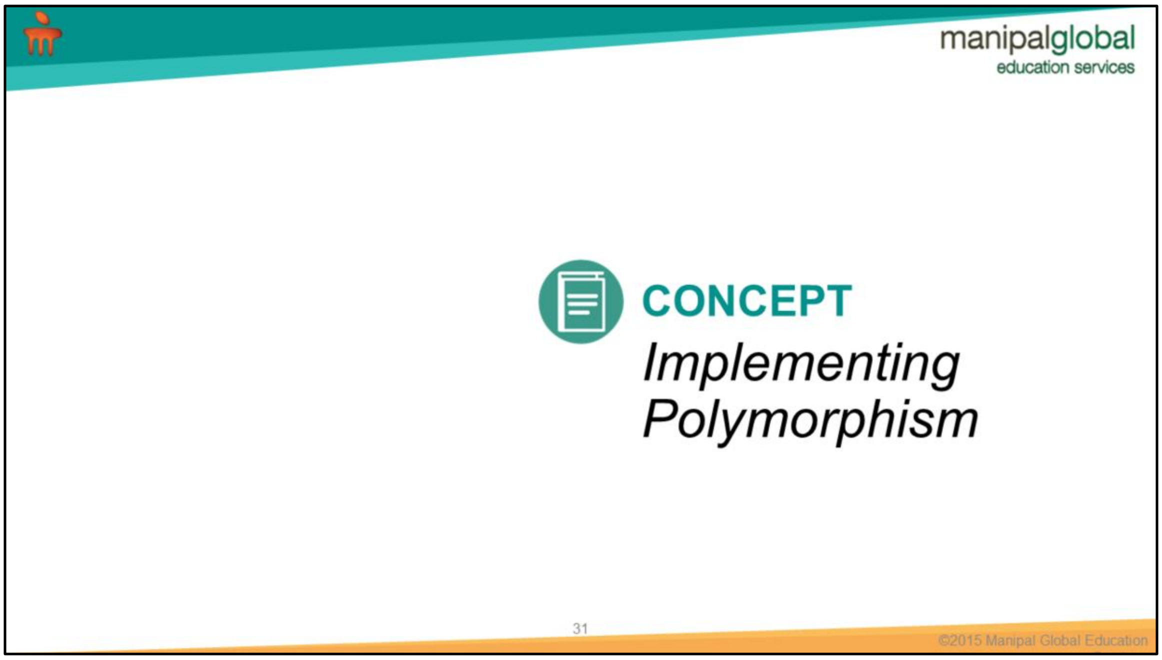


manipalglobal
education services

1. toString() method returns.

- ❑ String object
- ❑ StringBuffer object
- ❑ StringBuilder object
- ❑ Character object

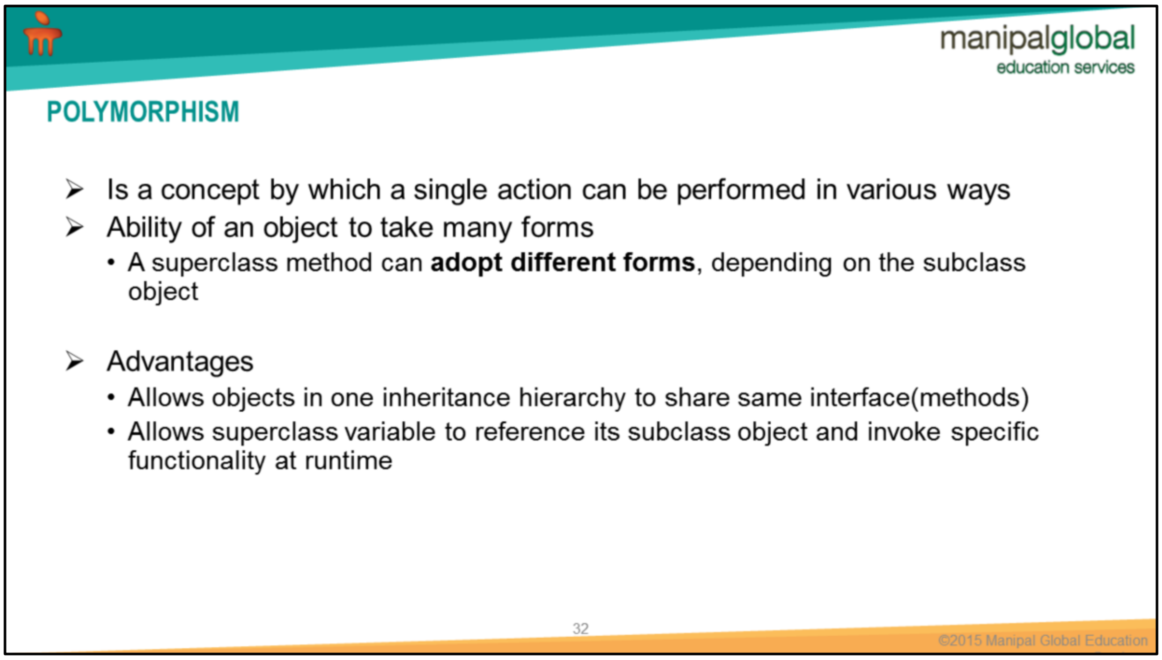
30



manipalglobal
education services

CONCEPT
Implementing Polymorphism

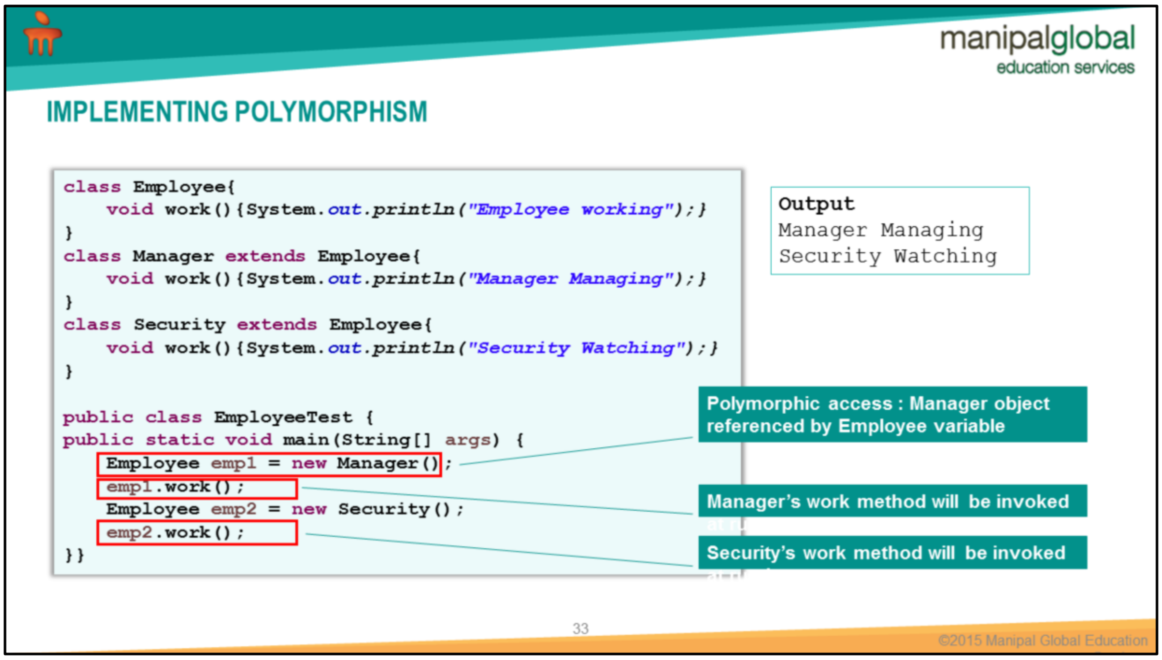
31



POLYMORPHISM

- Is a concept by which a single action can be performed in various ways
- Ability of an object to take many forms
  - A superclass method can **adopt different forms**, depending on the subclass object

- Advantages
  - Allows objects in one inheritance hierarchy to share same interface(methods)
  - Allows superclass variable to reference its subclass object and invoke specific functionality at runtime



manipalglobal
education services

IMPLEMENTING POLYMORPHISM

```
class Employee{
    void work(){System.out.println("Employee working");}
}
class Manager extends Employee{
    void work(){System.out.println("Manager Managing");}
}
class Security extends Employee{
    void work(){System.out.println("Security Watching");}
}

public class EmployeeTest {
public static void main(String[] args) {
    Employee emp1 = new Manager();
    emp1.work();
    Employee emp2 = new Security();
    emp2.work();
}}
```

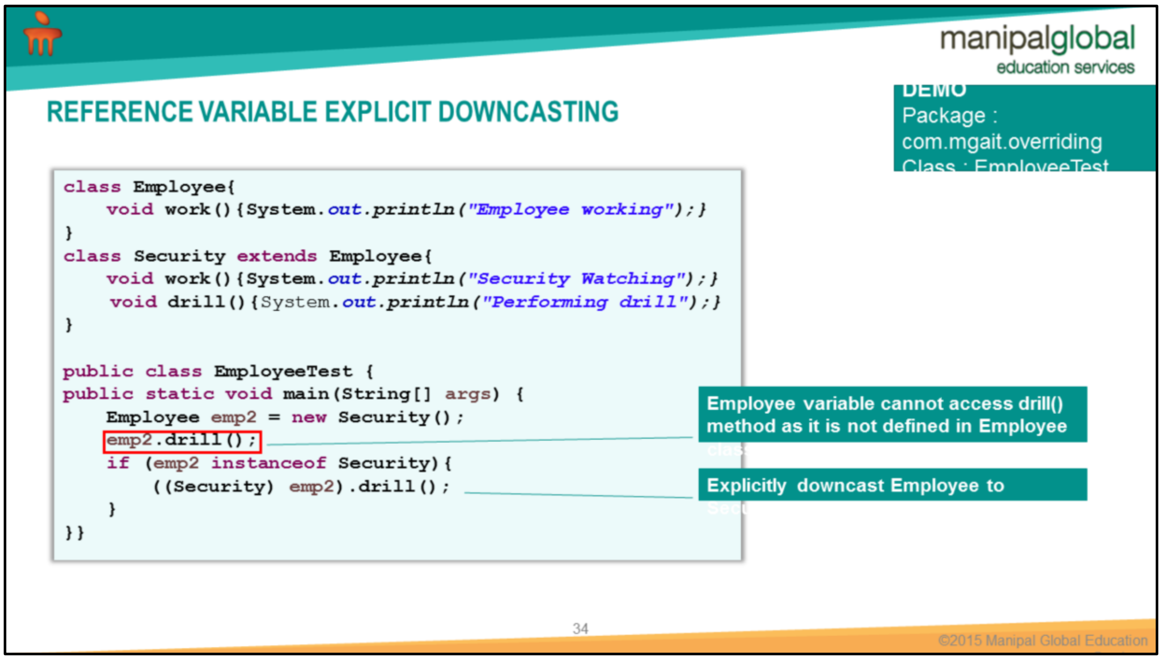
Output
```
Manager Managing
Security Watching
```

Polymorphic access : Manager object referenced by Employee variable

Manager's work method will be invoked

Security's work method will be invoked

33



manipalglobal
education services

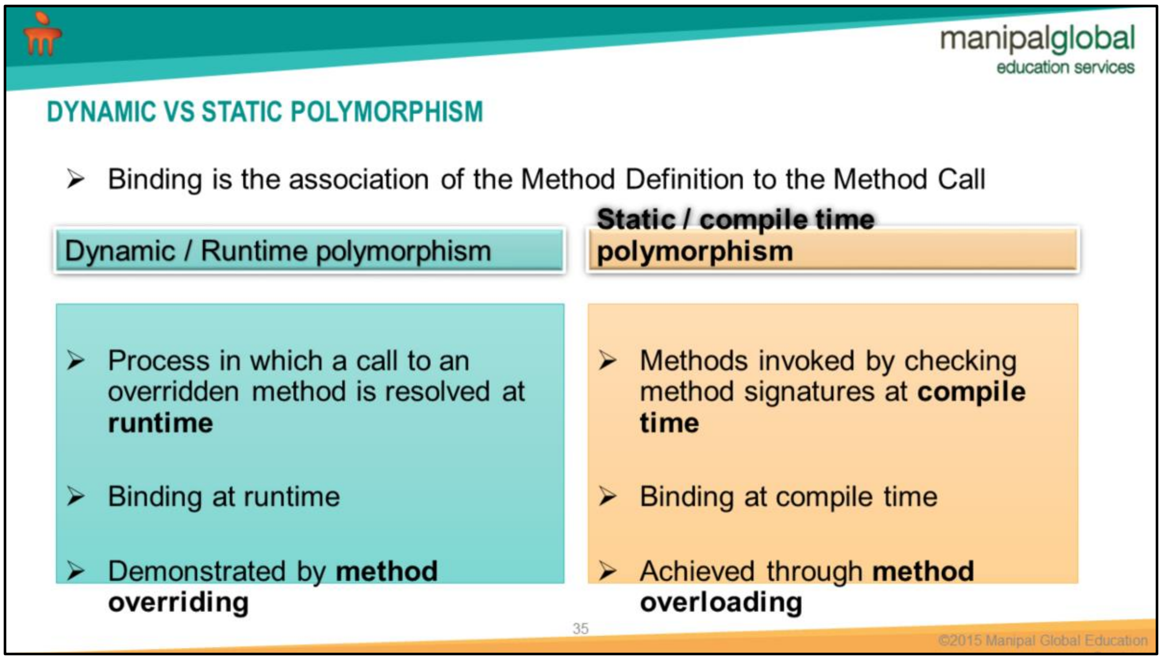
REFERENCE VARIABLE EXPLICIT DOWNCASTING

DEMO
Package :
com.mgait.overriding
Class : EmployeeTest

```
class Employee{
    void work(){System.out.println("Employee working");}
}
class Security extends Employee{
    void work(){System.out.println("Security Watching");}
     void drill(){System.out.println("Performing drill");}
}

public class EmployeeTest {
public static void main(String[] args) {
    Employee emp2 = new Security();
    emp2.drill();
    if (emp2 instanceof Security){
        ((Security) emp2).drill();
    }
}}
```

Employee variable cannot access drill() method as it is not defined in Employee

Explicitly downcast Employee to



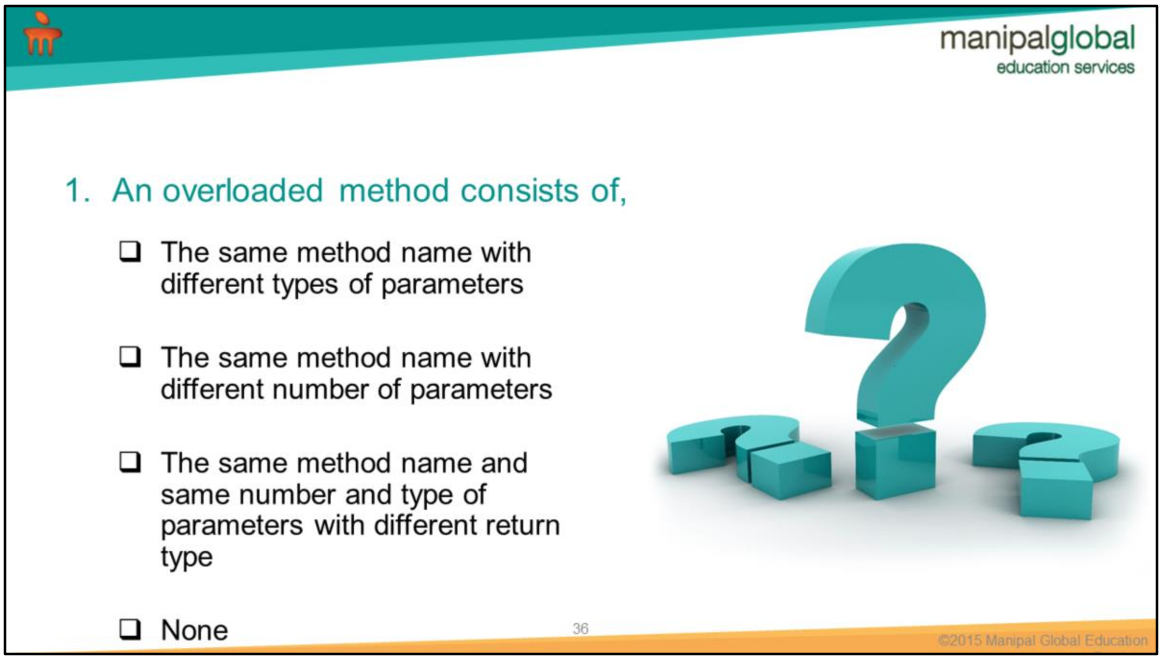
manipalglobal
education services

DYNAMIC VS STATIC POLYMORPHISM

- Binding is the association of the Method Definition to the Method Call

Dynamic / Runtime polymorphism	**Static / compile time polymorphism**
Process in which a call to an overridden method is resolved at **runtime**	Methods invoked by checking method signatures at **compile time**
Binding at runtime	Binding at compile time
Demonstrated by **method overriding**	Achieved through **method overloading**

35

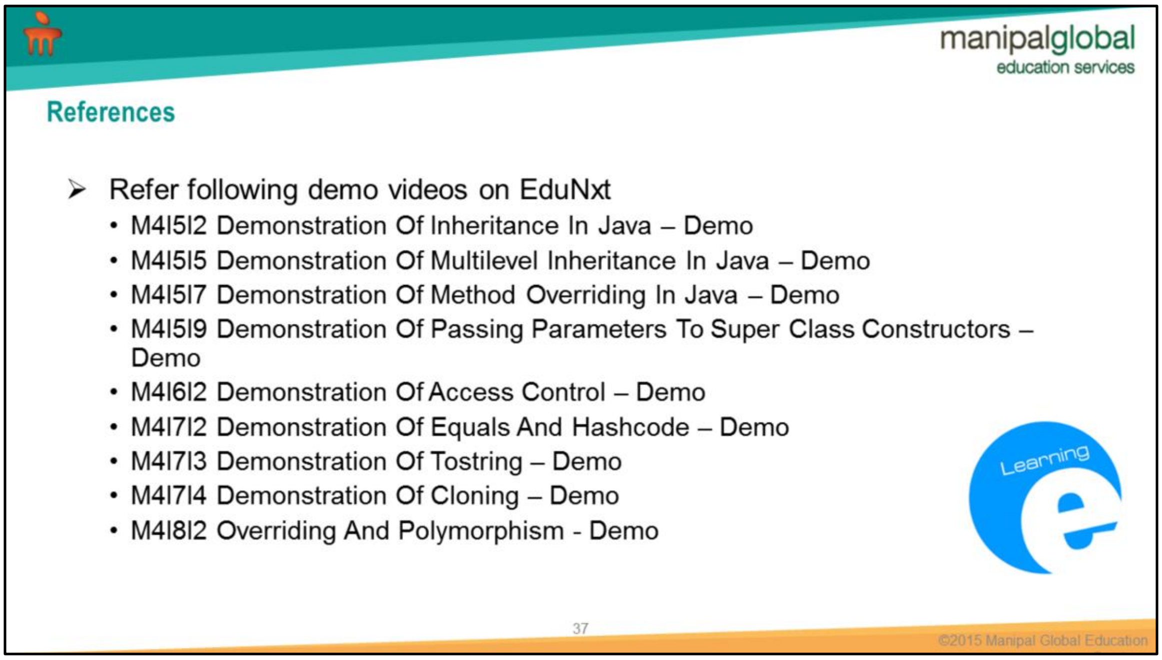


manipalglobal
education services

1. An overloaded method consists of,

- ❑ The same method name with different types of parameters
- ❑ The same method name with different number of parameters
- ❑ The same method name and same number and type of parameters with different return type
- ❑ None

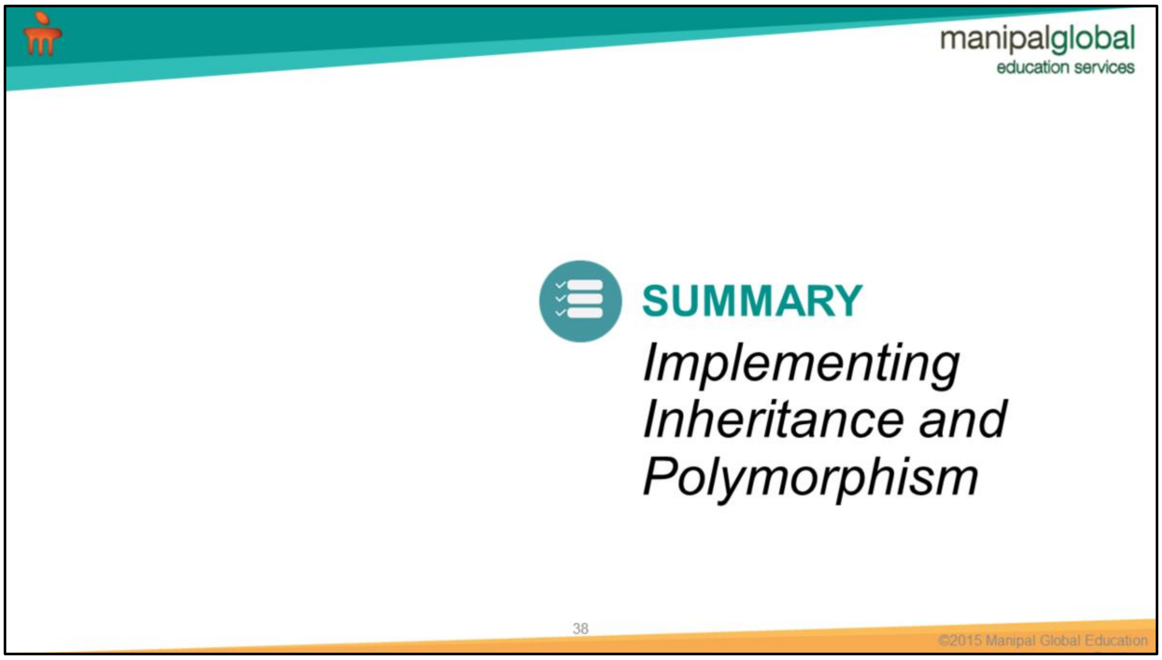
36



manipalglobal
education services

References

- Refer following demo videos on EduNxt
  - M4I5I2 Demonstration Of Inheritance In Java – Demo
  - M4I5I5 Demonstration Of Multilevel Inheritance In Java – Demo
  - M4I5I7 Demonstration Of Method Overriding In Java – Demo
  - M4I5I9 Demonstration Of Passing Parameters To Super Class Constructors – Demo
  - M4I6I2 Demonstration Of Access Control – Demo
  - M4I7I2 Demonstration Of Equals And Hashcode – Demo
  - M4I7I3 Demonstration Of Tostring – Demo
  - M4I7I4 Demonstration Of Cloning – Demo
  - M4I8I2 Overriding And Polymorphism - Demo



manipalglobal
education services

SUMMARY

Implementing Inheritance and Polymorphism

SUMMARY

In this lesson, you've learned to:

- Implement Inheritance between classes
- Use methods of cosmic classes.
- Demonstrate method overriding
- Implement Polymorphism