BITS Pilani

Object-Oriented Programming (CS F213)

Module III: Inheritance and Polymorphism in Java

CS F213 RL 11.3: Type Inquiry

Dr. Pankaj Vyas
Department of Computer Science, BITS-Pilani, Pilani Campus

CS F213 RL 11.3 : Topics

- Type Inquiry

Type Inquiry

1. instanceof operator tests whether the type of an object reference is a subtype of given type or not.

2. Syntax :

   if(e instanceof S) { .... }

   $$\begin{tikzcd}\text{e} \arrow[d, leftarrow] & \text{S} \arrow[d, leftarrow] \\ \text{Object reference} & \text{Type [May be a class Type or an interface]}\end{tikzcd}$$

3. The above statement tests whether e is a instance of type S or not.

4. This statement returns true if only if

   e is a direct instance of S or e belongs to one of the sub classes of S.

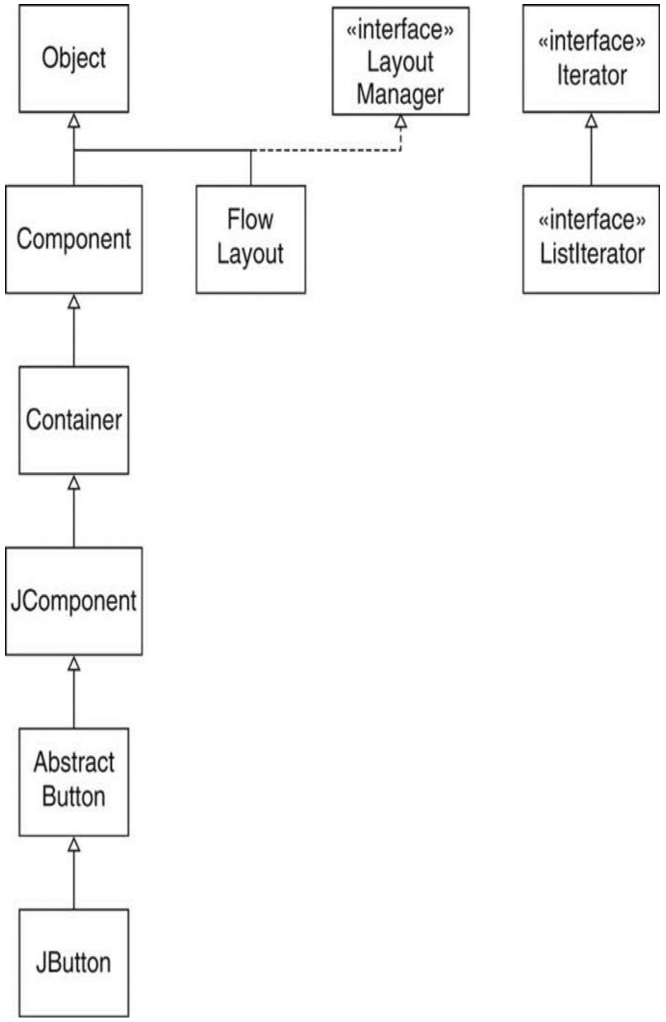
5. instanceof operator tests whether a reference value is a subtype of a given type or not. **But it does not give the exact type to which e belongs.**

6. If e is null then instanceof does not throw an Exception but simply returns false.

Type Inquiry : Example

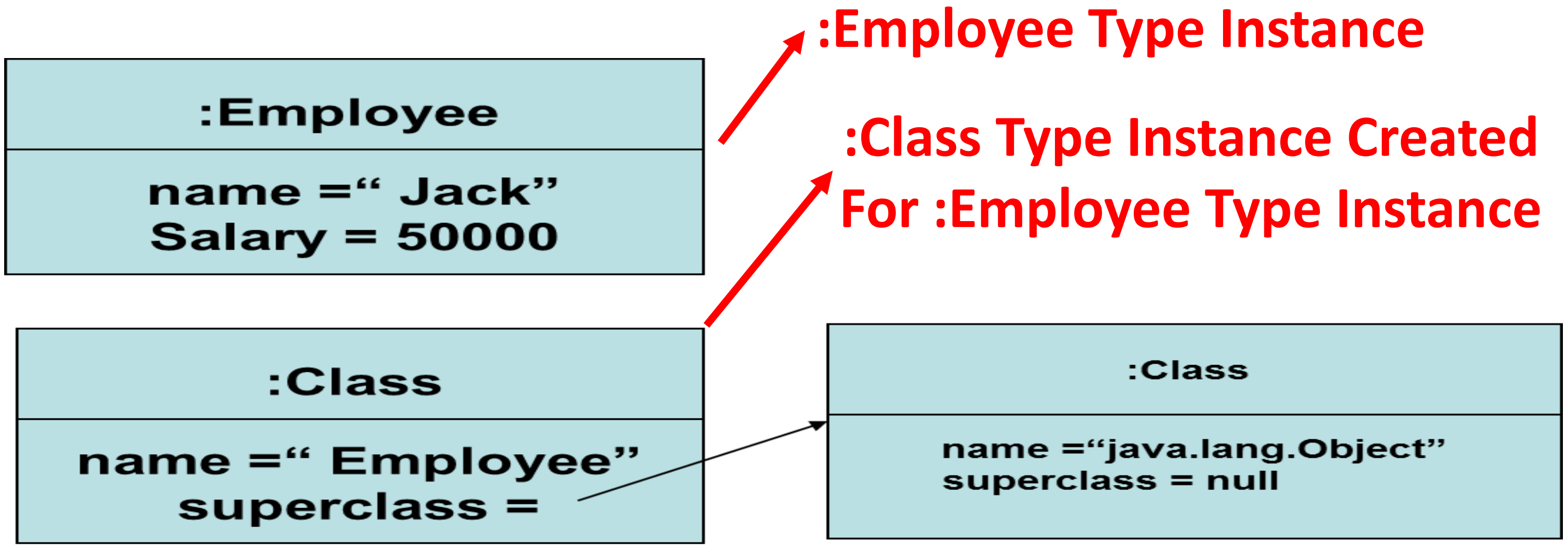
JButton b = new JButton("Hello");	
if (b instanceof Object)	returns true
if (b instance of Iterator)	returns false

Container c = new Container();	
if (c instanceof Component)	returns true
if (c instance of LayoutManager)	returns true



Role of java.lang.Class in Java

- Java creates an instance of Class (java.lang.Class) type for each object instantiation
- An instance of type Class is a type descriptor. It contains information about a given type such as class-name and super-class-name.
- Assume a class named 'Employee' with instance fields name:String and salary:double



How to get an Instance of :Class Type?

- Given an object reference of any type, we can get a java.lang.Class type instance using getClass() method. Suppose 'e' is any instance then

```
Class c = e.getClass();
```

- getClass() method returns a Class Type Object. Once you have a class object its name can be retrieved or printed as follows

```
System.out.println(e.getClass().getName());
```

- Adding a suffix .class to a Type also yields a Class Type object.

```
Rectangle.class, Employee.class , Student.class
```

How to know the Exact class of an instance?

1. Adding a suffix .class to a class name always yields a Class Object.
2. To test whether 'std' is a reference belonging to class Student or not use

```
if (std.getClass() == Student.class)
```

3. To test whether 'emp' is a reference of Employee class object or not

```
if(emp.getClass() == Emplyoee.class)
```

4. What about Arrays ?

```
BOX[ ] box = new BOX[5];
Class c = box.getClass();
if( c.isArray())
  System.out.println(" Component Type :"+ c.getComponentType());
```

Type Inquiry : Example

```
// FileName: TypeTest.java
class TypeTest
{
        public static void main(String args[])
        {
                String str = new String("Object");
                System.out.println(str.getClass().getName());

                // Checking whether str belongs to Object
                if(str instanceof Object)
                        System.out.println("Hello");
                else
                        System.out.println("Hi");

                // Checking whether str belongs to String
                if(str instanceof String)
                        System.out.println("Hello");
                else
                        System.out.println("Hi");
```

Type Inquiry : Example …

```
                if(str.getClass() == String.class)
                        System.out.println("Hello");
                else
                        System.out.println("Hi");
        }// End of Method
}// End of Class
```

Revisiting Object class in Java

- **Common super class for all other java classes.**
- **A class which is defined without extends clause is a direct sub class of Object class.**
- **Methods of Object class applies to all Java Objects.**
- **Important Methods:**
  - ❑**public boolean equals(Object other)**
  - ❑**protected Object clone()**
  - ✓**public String toString()**
  - ✓**public int hashCode()**

public Boolean equals(Object other)

- equals() method is actually used to test whether two objects have equal contents (states) or not.
- Type of Implicit parameter (this) and argument 'other' should be same, Otherwise false should be returned
- equals() method must be *reflexive*, *symmetric* and *transitive.*
- x.equals(x) should return true. (Reflexive)
- if x.equals(y) returns true then y.equals(x) should also return true (Symmetric)
- If x.equals(y) returns true and y.equals(z) returns true then x.equals(z) should also return true. (Transitive)
- For any non-null reference, x.equals(null) should return false.

public Boolean equals(Object other) : Example

```
// File Name :EqualsTest.java
class Student
{
        private String name;
        private String idno;
                // Assume Suitable Parametrized Constructor
                // Assume Accessor Methods
        public boolean equals(Object other)
        {
                if(other == null)                           return false;
                if(this.getClass() != other.getClass())     return false;
                if(this == other)                           return true;
                // Now Supply The Logic
                Student std = (Student) other;    // Type Cast
                boolean b1 = name.equals(other.getName())
                boolean b2 = idno.equals(other.getIdno())
                if(b1 && b2) return true;
                return false;
        }// End of Method
}// End of class
```

public boolean equals(Object other) : Example …

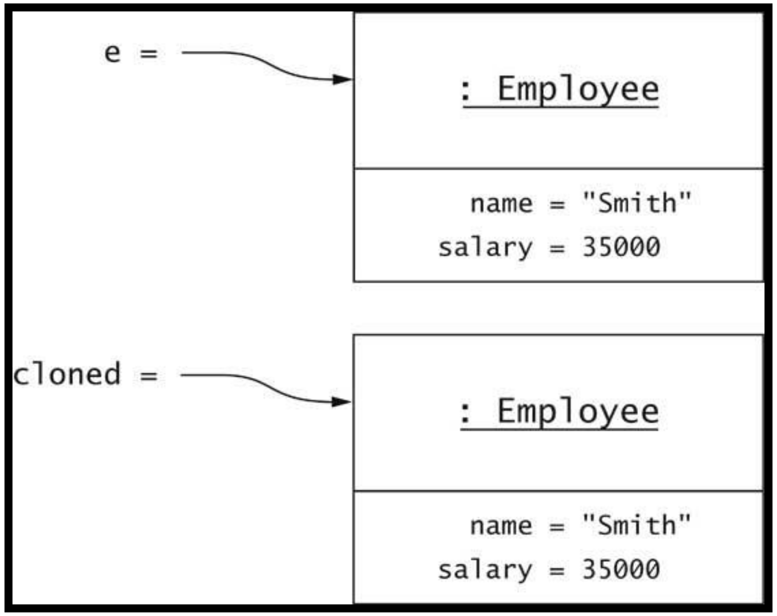
```
class HostlerStudent extends Student
{
        private int         hostelCode;
        private String      hostelName;
        private int         roomNo;
                // Assume Parameterized Constructor and Accessor Methods
        public boolean equals(Object other)
        {
                if(other == null) return false;
                if(this.getClass() != other.getClass()) return false;
                if(this == other) return true;
                Student std = (Student) other;
                if(!super.equals(std)) return false;
                HostlerStudent hstd = (HostlerStudent) other;
                boolean b1 = hostelCode == other.getHostelCode();
                boolean b2 = roomNo == other.getRoomNo();
                if(b1 && b2) return true;
                return false;
        } // End of Method
}// End of Class
```

protected Object clone()

- Clone of an object/instance is a separate instance but having equal states (contents)
- clone() method is used to create a clone of an object

Employee e = new Employee(…..);

Employee cloned = (Employee) e.clone();



Assumption :

Employee class supplies a suitable clone() method

Cloning Requirements and Conditions

- Clone of an object should be a new object but its state should be equal to its base object
- Cloning Conditions
  1. x.clone() != x
  2. x.clone().equals(x) return true
  3. x.clone().getClass() == x.getClass()
- Cloning Requirements
  - ➢ Any class willing to be cloned must
  1. Declare the clone() method to be public
  2. Implement an interface named 'Cloneable' [Note : Cloneable is a Tagging Interface. An Interface is a Tagging Interface if it does not have any method]

Cloning Example

```
class Employee implements Cloneable
{
    public Object clone()
    {
      try
      {
            super.clone();
      }
      catch(CloneNotSupportedException e){}
    }// End of Method
}// End of class
```

Condition 1: Class Must Implement Cloneable Interface

Condition 2: Class Must Override clone() method with public scope

Shallow Cloning

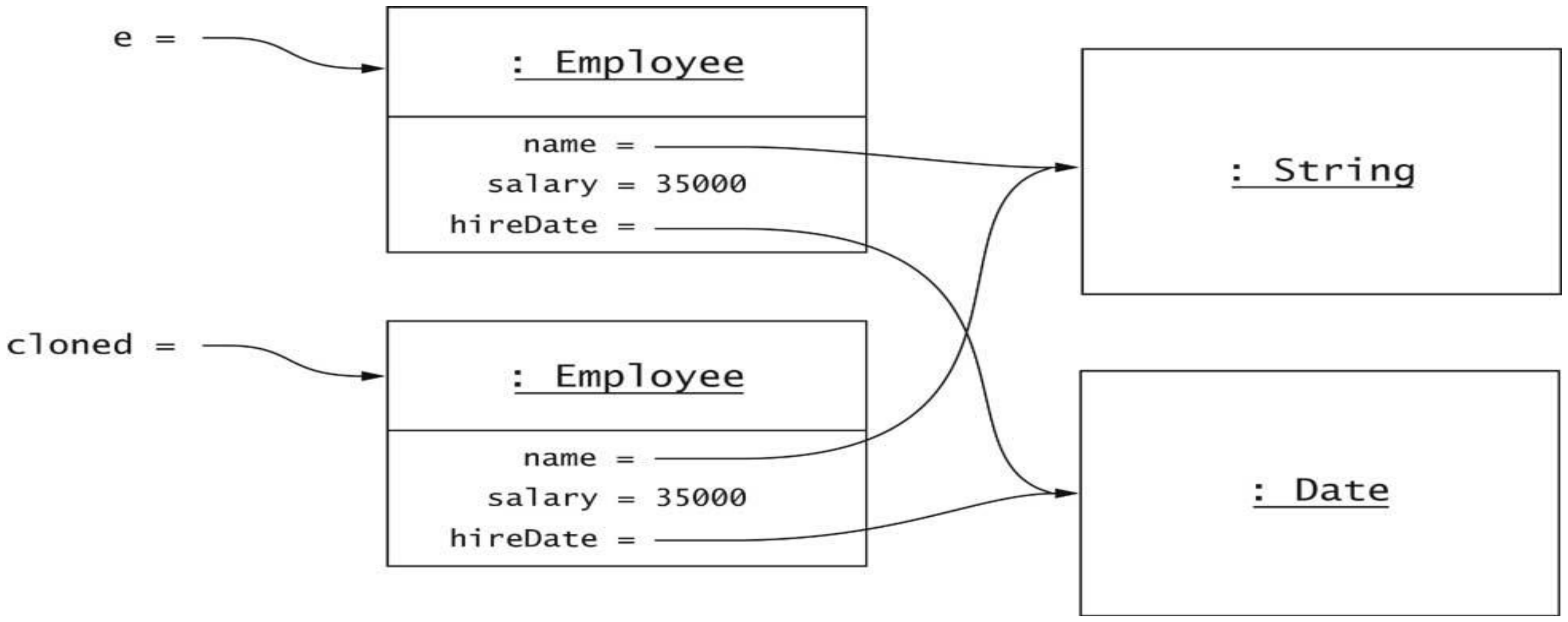
- clone() method makes a new object of the same type as the original and copies all fields.
- But, if the instance fields are object references of some other type then original and clone can share these references.

```
class Employee implements Cloneable
{
        private String   name;
        private double  salary;
        private Date     hire_date;
        ……….
        public Object clone()
        {
                // Assume Implementation
        }// End of Method
}// End of class
```

Shallow Cloning …

Employee e = new Employe(……);
Employee cloned = (Employee) e.clone();

Shallow Clone



Deep Cloning

- When a clone is allocated a separate space for every instance fields (except immutable instance field) then clone is called deep cloning

```
class Employee implements Cloneable
{
        public Object clone()
        {
                try
                {




                }
                catch(CloneNotSupportedException e) {return null;}
        } // End of Method
        ...
}// End of class
```

Deep Cloning

- When a clone is allocated a separate space for every instance fields (except immutable instance field) then clone is called deep cloning

```
class Employee implements Cloneable
{
        public Object clone()
        {
                try
                {
                        Employee cloned = (Employee)super.clone();



                }
                catch(CloneNotSupportedException e) {return null;}
        } // End of Method
        ...
}// End of class
```

Deep Cloning

- When a clone is allocated a separate space for every instance fields (except immutable instance field) then clone is called deep cloning

```
class Employee implements Cloneable
{
        public Object clone()
        {
                try
                {
                        Employee cloned = (Employee)super.clone();
                        cloned.hireDate = (Date)hiredate.clone();


                }
                catch(CloneNotSupportedException e) {return null;}
        } // End of Method
        ...
}// End of class
```

Deep Cloning

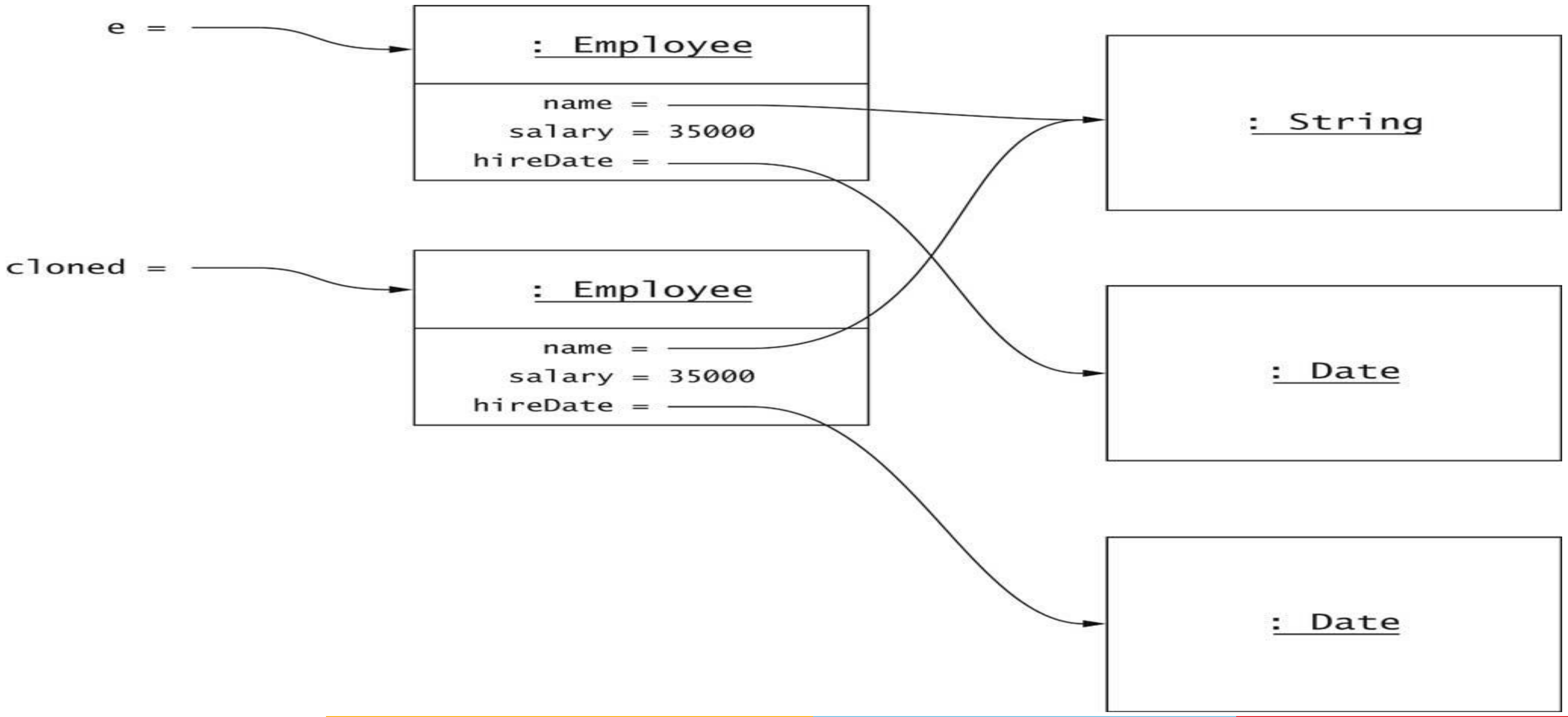
- When a clone is allocated a separate space for every instance fields (except immutable instance field) then clone is called deep cloning

```
class Employee implements Cloneable
{
        public Object clone()
        {
                try
                {
                        Employee cloned = (Employee)super.clone();
                        cloned.hireDate = (Date)hiredate.clone();
                        return cloned;
                }
                catch(CloneNotSupportedException e) {return null;}
        } // End of Method
        ...
}// End of class
```

Deep Cloning

Employee e = new Employe(……);
Employee cloned = (Employee) e.clone();

Deep Clone



innovate achieve lead

Thank You