

# Java Type System

What is type in Java?

Set of **Values** with a set of **Operations** that can be applied to the values.

**Example :**

**int type**      **->** Set of values {  $[-2^{31}]$  to  $[+2^{31}-1]$  }

Operations : All Arithmetic Operations

**BOX type**      **->** Set of all BOX object references

Operations : area() , volume() , toString()

Java is Strongly Typed Language.

Type checking is done both at Compile and Run Time.

- If a Type Check Fails at Compile Time **->** Compile Time Error
- If a Type Check Fails at Run Time **->** RunTime Exception

# Types in Java

1. *Primitive Type( int, float, double, byte, char, boolean, short, long)*
2. *A class Type [BOX, String, Arrays , Student etc]*
3. *An interface Type [ Either User Defined OR Library interfaces such as Comparable]*
4. *An array type*
5. *null type*

## *Note :*

1. *Arrays have a component type (Type of array elements) e.g String[] array has component type as String; int[] array has Component type as int*
2. *void is not a type in java. [only a keyword to denote that a method does not return anything]*
3. *Object[] names = new String[10]; What's Array Type and its component type.*

# Values in Java

1. A Value of Primitive type
2. A reference to an object
3. reference to an array
4. Null

## Examples :

<code>10 , 10.35 , true</code>	—————→	<b>Primitive Type Values</b>
<code>new BOX(10,6,8);</code>	—————→	<b>Reference to an object</b>
<code>new int[] { 10,6,8}</code>	—————→	<b>Reference to array</b>
<code>null</code>		

**Note :** You can not have a value of type interface.

# Nonprimitive types

	<b>type</b>	<b>object</b>
class types	Rectangle	<code>new Rectangle(2, 4, 8, 8)</code>
	String	<code>"dProg2"</code>
interface types	Shape	
	Comparable	
array types	<code>int[][]</code>	<code>new int[3][7]</code>
	<code>String[]</code>	<code>{"dIntProg", "dProg2"}</code>

- no objects of interface type!

<b>type</b>	<b>value</b>
null type	<code>null</code>

# QUIZ

static/dynamic type

Which – if any – errors arise?

Employee
setSalary()

No error	Compiler error	Exception on runtime
----------	----------------	----------------------

- |     |              |       |       |
|-----|--------------|-------|-------|
| 1.  | a),b)        |       |       |
| 2.  | a)           | b)    |       |
| 3.  | a)           |       | b)    |
| 4.  | b)           | a)    |       |
| 5.  |              | a),b) |       |
| 6.  |              | a)    | b)    |
| 7.  | b)           |       | a)    |
| 8.  |              | b)    | a)    |
| 9.  |              |       | a),b) |
| 10. | I don't know |       |       |

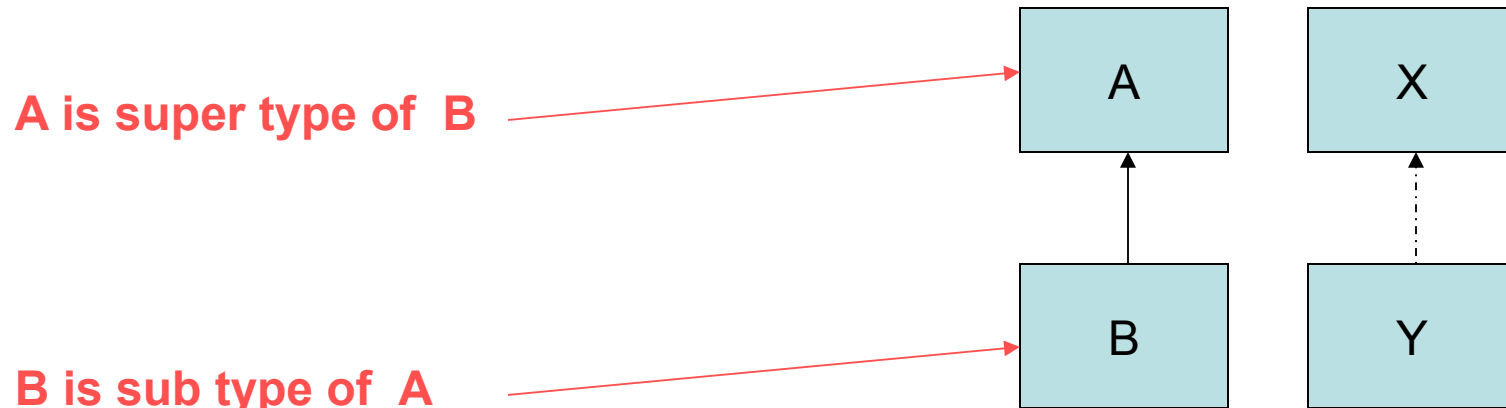
Employee e = null;

a) e.clear();

b) e.setSalary(1000);

# Sub Types

- Sub type specifies the inheritance relationship either by extending a class or implementing an interface

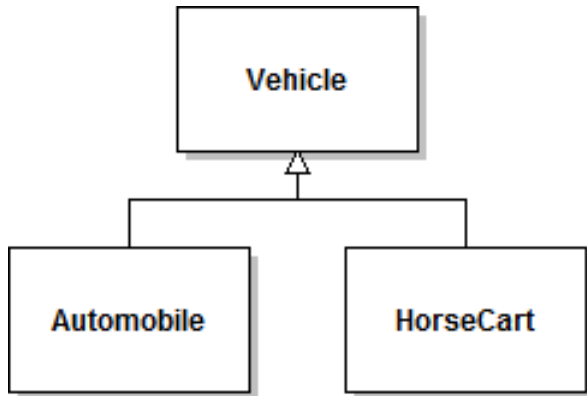


Similarly X is super type for Y and Y is sub type for X.

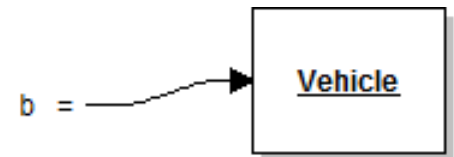
*You can substitute a value of subtype whenever supertype value is expected*

# Non-primitive type: variables

- Variables of non-primitive type contains reference to object of **same** type

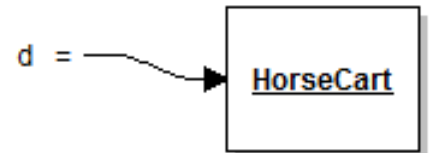
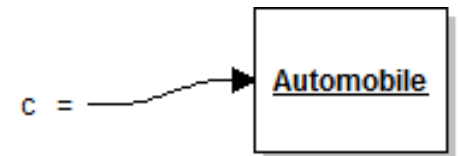


```
Vehicle b;  
b = new Vehicle();
```



- or of a **subtype**

```
Vehicle c;  
c = New Automobile();  
Vehicle d;  
d = new HorseCart();
```



# Example

**X    x1    =    ?**

**What's Expected**



**x1 is reference variable of type X?**

**If X is an interface**

**RHS can be an instance of any class implementing X.**

**If X is abstract class**

**RHS can be an instance of any concrete subclass of X**

**If X is a concrete class**

**RHS can be either an instance of X or any of its subclasses**

# Rules for Subtype Relationships

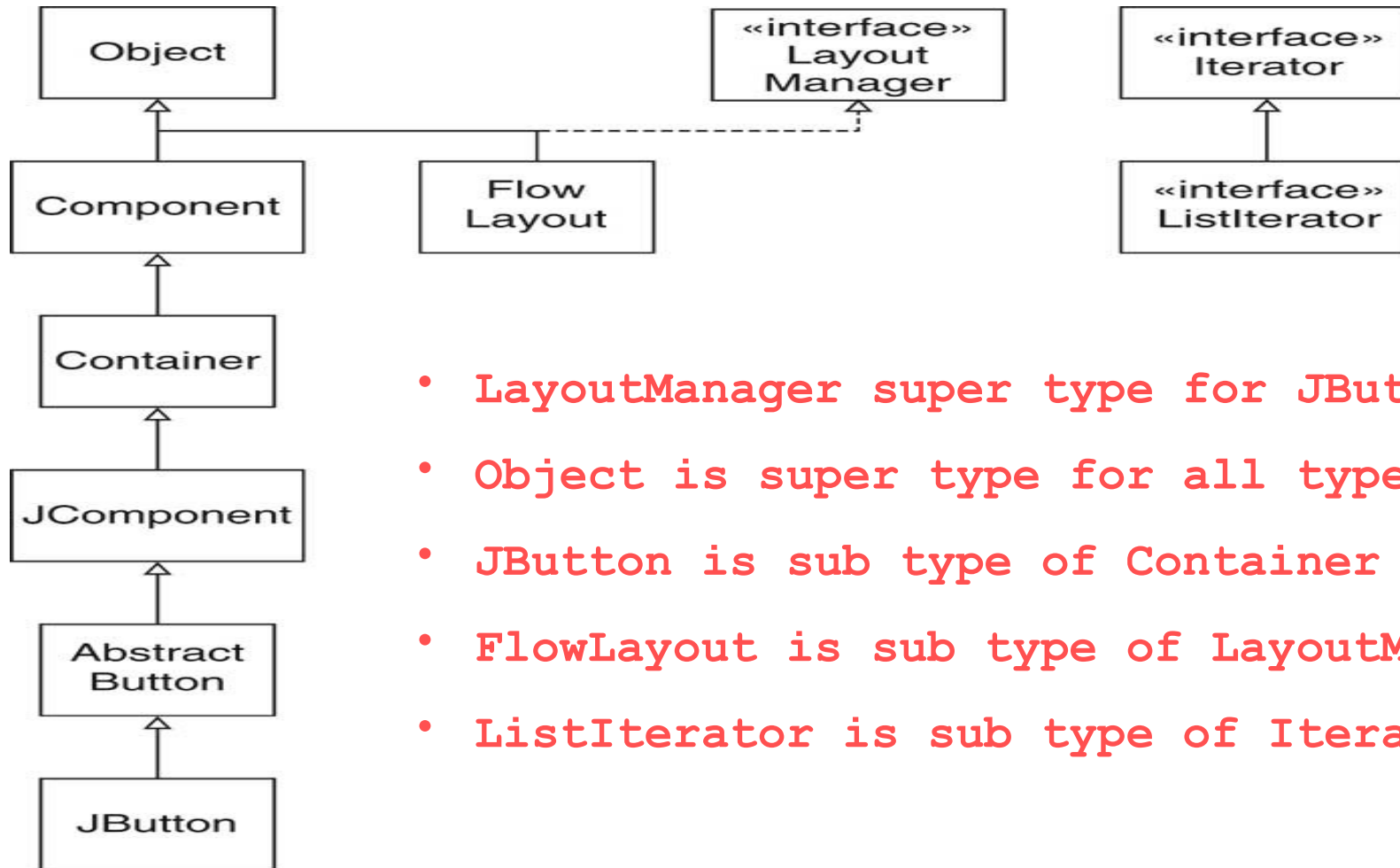
*S is a subtype of T if*

1. S and T are the same type
2. S and T are both class types, and T is a direct or indirect superclass of S
3. S is a class type, T is an interface type, and S or one of its superclasses implements T
4. S and T are both interface types, and T is a direct or indirect superinterface of S
5. S and T are both array types, and the component type of S is a subtype of the component type of T
6. S is not a primitive type and T is the type Object
7. S is an array type and T is Cloneable or Serializable
8. S is the null type and T is not a primitive type

# Subtype Relationship

- T1 is subtype of T2,  $T1 \leq T2$ 
  - if T1 is the same type as T2
  - or T1 implements T2
  - or T1 extends T2
  - or there is T3 such that  $T1 \leq T3$  and  $T3 \leq T2$
  - or T1 is T3[], T2 is T4[] and  $T3 \leq T4$
  - or T1 is array type and T2 is `Cloneable` or `Serializable`
  - or T1 is non-primitive and T2 is `Object`
  - or T1 is null type and T2 is non-primitive

# Examples



- **LayoutManager** super type for **JButton**
- **Object** is super type for all types
- **JButton** is sub type of **Container**
- **FlowLayout** is sub type of **LayoutManager**
- **ListIterator** is sub type of **Iterator**

# Examples continued ...

1. Is Container is a subtype of Component ? *TRUE*
2. Is JButton is a subtype of Component ? *TRUE*
3. Is FlowLayout is a subtype of LayoutManager? *TRUE*
4. Is ListIterator is a subtype of Iterator ? *TRUE*
5. Is Rectangle[ ] is a subtype of Shape[ ] ? *TRUE*
6. Is int[ ] is a subtype of Object ? *TRUE*
7. Is int is subtype of long ? *NO*
8. Is long is a subtype of int ? *NO*
9. Is int[ ] is a subtype of Object[ ] ? *NO*
10. Is int[] is a subtype of Object ? *YES*
11. Iterator is a subtype of Object *YES*

**Primitive Types are not implemented as  
Objects**

# QUIZ

Yes/No

- 1 Comparable is a subtype of Object?
- 2 Comparable is a subtype of String?
- 3 String is a subtype of Comparable?
- 4 int[] is a subtype of Object?
- 5 int[] is a subtype of Object[]?
- 6 Serializable[] is a subtype of Object[]?
- 7 int[][] is a subtype of Serializable[]?

# Type check: compile time

- Static versus dynamic type

```
Vehicle v;
```

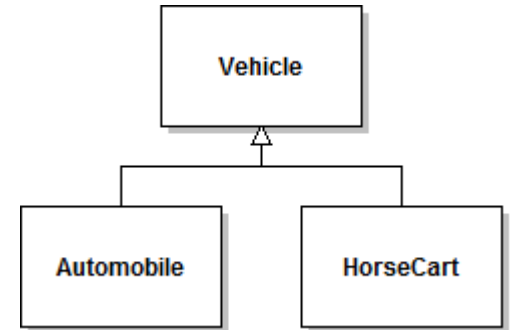
```
    v = new Automobile();
```

```
Object obj;
```

```
    obj = v;
```

**Static** type of v is Vehicle

**Dynamic** type of v is Automobile



- Compiler looks at **static** type only

```
Automobile bmw
```

```
    bmw = v;
```

compile time error

- Use **type cast**

```
    bmw = (Automobile) v;
```

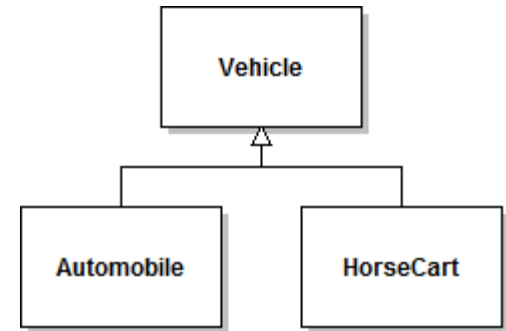
**Static** type of (Automobile) v is Automobile

- Compiler is happy!

# Type check: runtime

- You may fool the compiler:

```
Vehicle v = new HorseCart();  
Automobile bmw = (Automobile) v;
```



**Static** type of (Automobile)v is Automobile

- Compiler is happy
- But at runtime

Exception in thread "main"

java.lang.ClassCastException: HorseCart cannot be cast to Automobile

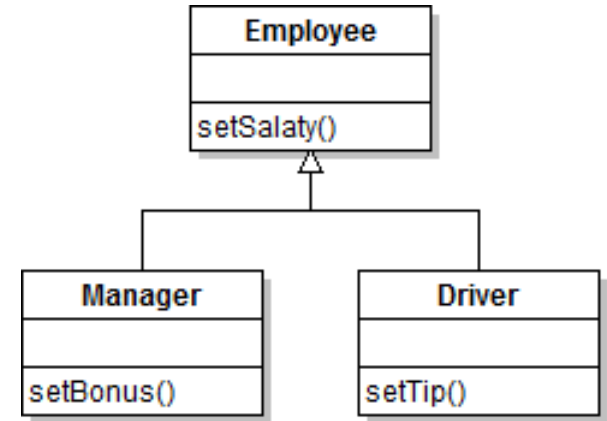
**Dynamic** type of v is HorseCart

# QUIZ

static/dynamic type

Which – if any – errors arise?

No error	Compiler error	Exception on runtime
----------	----------------	----------------------



- |     |              |       |       |
|-----|--------------|-------|-------|
| 1.  | a),b)        |       |       |
| 2.  | a)           | b)    |       |
| 3.  | a)           |       | b)    |
| 4.  | b)           | a)    |       |
| 5.  |              | a),b) |       |
| 6.  |              | a)    | b)    |
| 7.  | b)           |       | a)    |
| 8.  |              | b)    | a)    |
| 9.  |              |       | a),b) |
| 10. | I don't know |       |       |

```
Employee e = new Driver();
```

```
a) e.setBonus(10);
```

```
b) ((Manager)e).setBonus(10);
```

# The ArrayStoreException

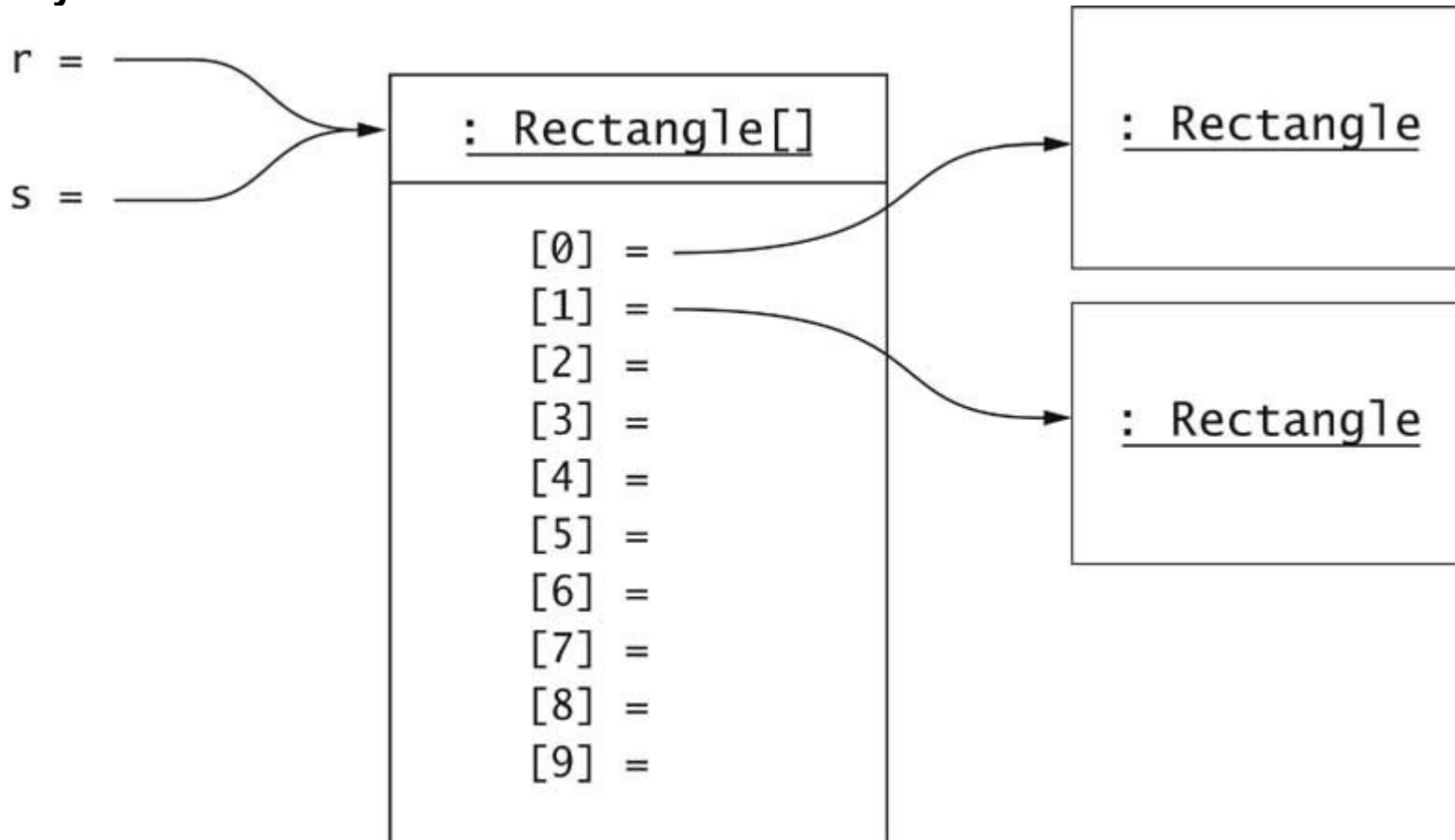
• **Rectangle[] is a subtype of Shape[] ?.**      **TRUE**

- Can assign Rectangle[] value to Shape[] variable:
- ```
Rectangle[] r = new Rectangle[10];  
Shape[] s = r;
```
- Both r and s are references to the same array
- The assignment  

```
s[0] = new Polygon();
```

 compiles  
But Throws an ArrayStoreException at runtime
- Each array remembers its component type

## Array References



# Wrapper Classes

1. Primitive types aren't Objects in Java.
2. We Can wrap primitive types in Objects using wrapper classes.
3. Wrapper class for each type are:

|           |       |        |         |
|-----------|-------|--------|---------|
| Integer   | Short | Long   | Byte    |
| Character | Float | Double | Boolean |


4. Auto-boxing and auto-unboxing

```
ArrayList<Integer> numbers = new ArrayList<Integer>();  
numbers.add(13);           // calls new Integer(13)  
int n = numbers.get(0);    // calls intValue();
```

5. Wrapper classes are immutable.

# Example

```
import java.util.*;
class Wraptest
{
public static void main(String args[])
{
ArrayList<Integer> ints = new ArrayList<Integer>();
ints.add(10);
ints.add(20);
ints.add(30);
ints.add(40);
```



```
ints.add(new Integer(10));
ints.add(new Integer(20));
ints.add(new Integer(30));
ints.add(new Integer(40));
```

```
for(int i=0;i<ints.size();i++)
System.out.println(ints.get(i));
}
}
```

*ints.get(i).intValue();*

**E:\loop>java Wraptest**

**10**

**20**

**30**

**40**

# Enumerated Types

1. Type with finite set of values

2. Example: `enum Size { SMALL, MEDIUM, LARGE }`  
*`public enum MaritalStatus { MARRIED, UNMARRIED }`*

3. Typical use:

`Size imageSize = Size.MEDIUM;`  
`if (imageSize == Size.SMALL) . . .`

4. Syntax :

*`accessSpecifier enum TypeName { value1, value2, .....value n }`*

5. Safer than integer constants

`public static final int SMALL = 1;`  
`public static final int MEDIUM = 2;`  
`public static final int LARGE = 3;`

# Type safe Enumerations

1. enum equivalent to class with fixed number of instances

```
public class Size
```

```
{
```

```
    private /* ! */ Size() { }
```

```
    public static final Size SMALL = new Size();
```

```
    public static final Size MEDIUM = new Size();
```

```
    public static final Size LARGE = new Size();
```

```
}
```

2. enum types are classes; can add methods, fields, constructors

# Define enum type Month

```
public class Month
{
    private String monthName;

    private Month(String month) { monthName = month; }

    public static final Month JAN = new Month("January");

    .....

    .....

    public static final Month DEC = new Month("December");
}
```

# 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){ ..... }
```



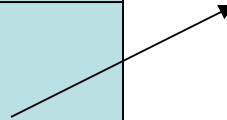
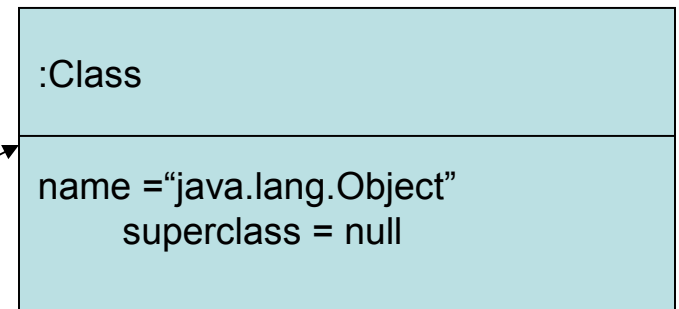
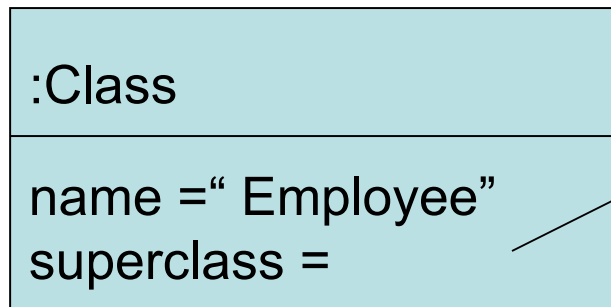
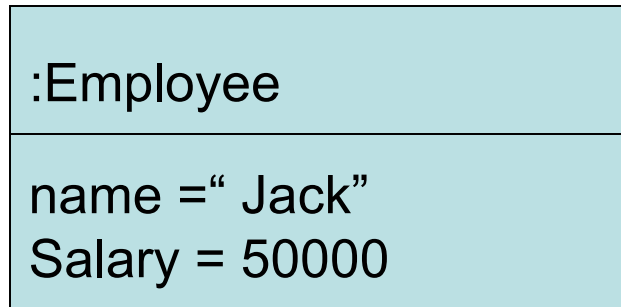
Object  
reference

Type [may be a class or interface]

3. The above statement tests whether e is a instance of type S or not.
4. This statement returns true if e is a direct instance of S or e belongs to one of the sub classes of S.
5. **Instanceof** operator can test whether the type of a value is 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.

# Class class

- A Class object is a type descriptor. It contains information about a given type such as type name and super class.



# Class class

- Given a object reference we can know the exact type of the object by using getClass() method

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

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

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

- Adding the suffix .class to a type also yields the Class object.

Rectangle.class, Employee.class , Student.class

# Knowing Exact class of Reference

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

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

3. To test whether emp is a refrence for 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())
```

```
S.O.P (" Component Type :"+ c.getComponentType());
```

```
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");
        if(str.getClass() == String.class)
            System.out.println("Hello");
        else
            System.out.println("Hi");
    }
}
```

```
E:\oop>java typetest
java.lang.String
Hello
Hello
Hello
```

```
class TypeTest
{
    public static void main(String args[])
    {
        String[] names = new String[5];
        Class c1 = names.getClass();
        System.out.println(c1.getName());
        // System.out.println(names.getClass().getName());
        if(c1.isArray())
        System.out.println(c1.getComponentType());
        Object[] objs = new Integer[10];
        System.out.println(objs.getClass().getName());
        System.out.println(objs.getClass().getComponentType());
    } // End of main()
} // End of class TypeTest
```

```
[Ljava.lang.String;
class java.lang.String
[Ljava.lang.Integer;
class java.lang.Integer
```

# ***Exercises***

Q7.1 Which Types can you use for variables but not for values?

Q7.2 What is the type null?

Q7.4 Write a Program that generates an ArrayStoreException?

Q7.5 When do you use wrapper classes for primitive types?

Q7.6 What Java code do you use to test

1. Whether x belongs to the Rectangle class

2. x belongs to a subclass of JPanel class

(But not the JPanel class itself)

3. Whether class of x implements Cloneable interface

Q7.7 Distinct ways of obtaining class Object that describes a Rectangle class

**Class c = java.awt.Rectangle.class;**

**Class c = new java.awt.Rectangle().getClass();**

**Class c = Class.forName("java.awt.Rectangle"); // can throw exception**

# Object class

- 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:

1. String toString()

2. boolean equals(Object other)

3. int hashCode()

4. Object clone()

# public String toString()

- Returns a string representation of the object
- Useful for debugging
- toString used by concatenation operator
- aString + anObject means aString + anObject.toString()
- User can override the toString() method.

```
class BOX
{
.....
public String toString()
{
.....
..... . .
}
}
```

```
class Student
{
.....
public String toString()
{
.....
..... . .
}
}
```

## **toString() continued...**

- **toString() is automatically called when you**
  - 1. concatenate an object with a string**
  - 2. print an object with print or println method**
  - 3. when you pass an object reference e to assert statement**
- **Default implementation of toString() method returns the name of the class and the hash code of the object.**

## ***Example***

```
class Student
{
private String name, idno;
Student(String name, String idno)
{
this.name = name; this.idno = idno;
}
public String toString()
{
return name+" "+idno;
}
}
```

```
class HostlerStudent extends Student
```

```
{
private int hostelCode;
private String hostelName;
private int roomNo;
HostlerStudent(String name, String idno, int hcode, String hname, int rno)
{
Super(name,idno); hostelCode = hcode; hostelName = hname ; roomNo = rno;
}
public String toString()
{
return super.toString()+" "+hostelName+" " + roomNo;
}
}
```

## **public boolean equals(Object other)**

- equals method tests whether two objects have equal contents or not.
- Equals method must be *reflexive*, *symmetric* and *transitive*.
- `x.equals(x)` should return true. (Reflexive)
- `x.equals(y)` returns true iff `y.equals(x)` returns true
- If `x.equals(y)` returns true and `y.equals(z)` returns true then `x.equals(z)` should also return true.
- For any non-null reference, `x.equals(null)` should return false.
- Users can either overload or override `equals()` method.

# equals() overloading examples

```
class BOX
{
.....
.....
.....
public boolean equals(BOX other)
{
<< Implement equals logic>>
.....
}
}
```

BOX Parameter



**OVERLOADING**

```
class Student
{
.....
.....
.....
public boolean equals(Student other)
{
<< Implement equals logic>>
.....
}
}
```

Student Parameter



# equals() overriding examples

## OVERRIDING

Object Type Parameter

```
class BOX
{
.....
.....
.....
public boolean equals(Object other)
{
BOX other = (BOX) other;
<< Implement equals logic>>
.....
}
}
```

```
class Student
{
.....
.....
public boolean equals(Object other)
{
Student other = (Student) other;
<< Implement equals logic>>
.....
}
}
```

```
class Student
{
private String name;
private String idno;
.....
.....
// 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;
```

```
Student std = (Student) other;
boolean b1 = name.equals(other.getName())
boolean b2 = idno.equals(other.getIdno())
if(b1 && b2) return true;
return false;
}
}
```

Cont...

```
class HostlerStudent extends Student
```

```
{
```

```
private int hostelCode;
```

```
private String hostelName;
```

```
private int roomNo;
```

```
.....
```

```
.....
```

```
// 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;
```

```
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;
```

```
}
```

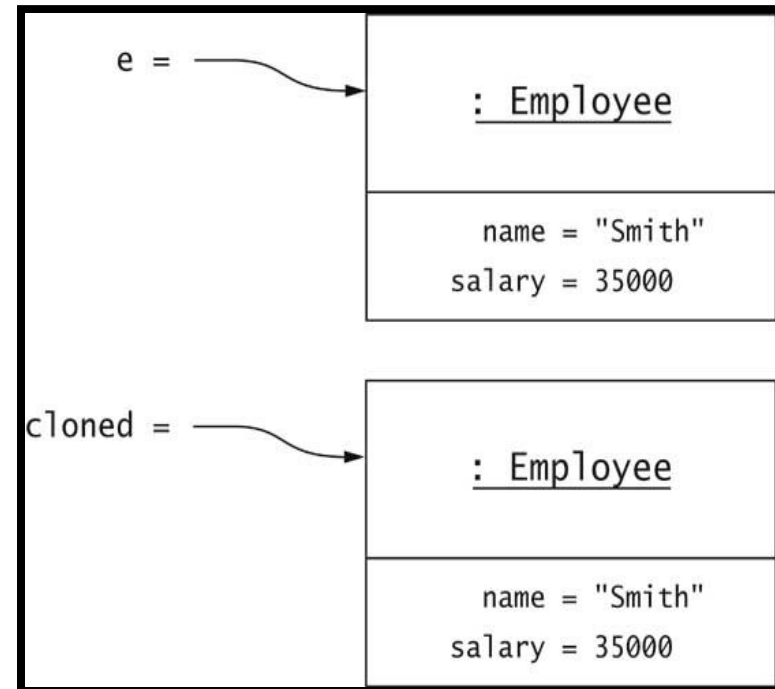
```
}
```

# Shallow and Deep copy

- **clone()** method is used to make the clone or deep of the object.
- **Example :**

Employee e = new Employee(.....);

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



Assumption :

Employee class supplies a suitable clone() method

# Cloning Conditions

- `x.clone() != x`
- `x.clone().equals(x)` return true
- `x.clone().getClass() == x.getClass()`

*“ clone should be a new object but it should be equals to its original”*

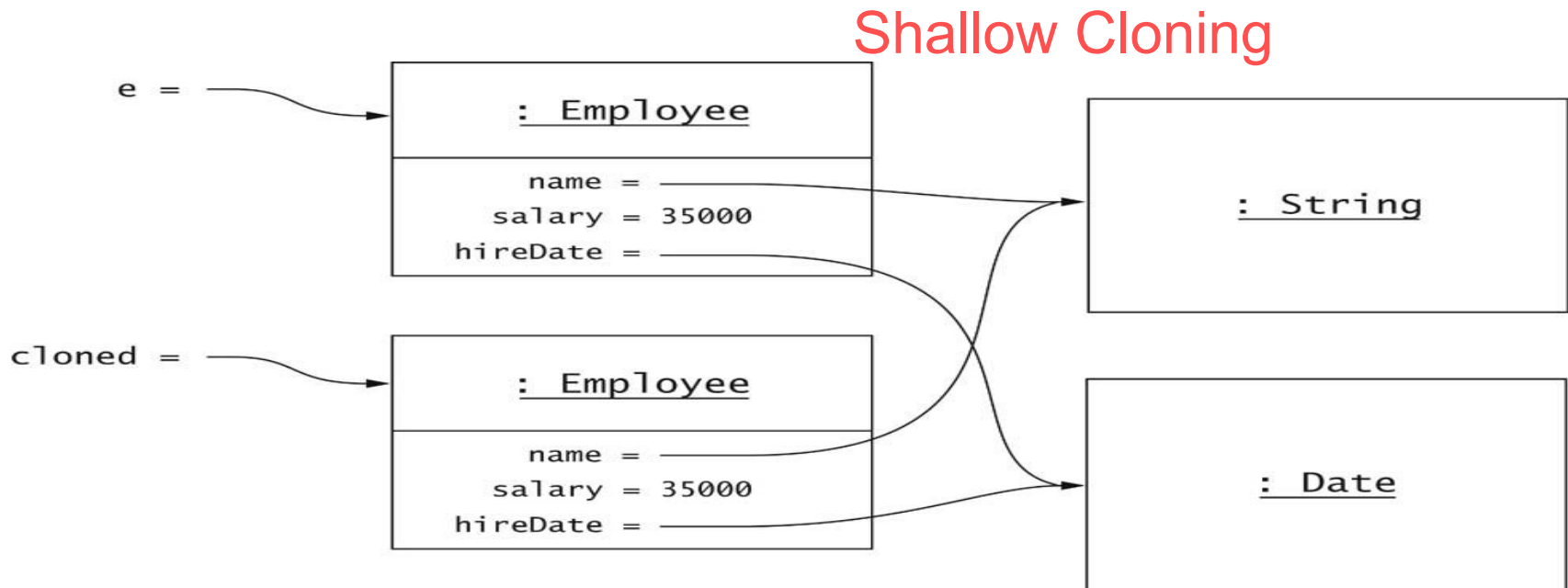
# clone requirements

- Any class willing to be cloned must
  1. Declare the clone() method to be public
  2. Implement Cloneable interface

```
class Employee implements Cloneable
{
    public Object clone()
    {
        try { super.clone() }
        catch(CloneNotSupportedException e){ .. }
    }
}
```

# Shallow Copy

- Clone() method makes a new object of the same type as the original and copies all fields.
- But if the fields are object references then original and clone can share common subobjects.



# Deep Cloning

```
public 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; // won't happen
        }
    }
    ...
}
```

