Over Riding:

Whatever methods parent has by default available to the child through inheritance. If Child class not satisfied with parent class implementation then child is allowed to redefine that method based on its requirement this process is called overriding.

Parent class method which is overridden is called overridden method and the child class method which is overriding is called overriding method.

Example:

Class P {

Public void property () {

Sop (“cash+gold+loan”);

}

Public void marry () {

Sop (“subba laxmi”);

}

}

Class C extends P {

Public void property () {

Sop (“cash+gold+loan”);

}

Public void marry () {

Sop (“3sha|Charmi”);

}

}

Class Test {

Public static void main (String [] args) {

P p = new P ();

p.marry (); // -> Parent class method

C c = new C ();

c .marry (); // -> Child class method

P pp = new C ();

pp. marry (); //-> Child class method

}

}

In overriding, Method resolution always takes care by jvm based on run time object and hence overriding is considered as runtime polymorphism or dynamic polymorphism or late binding.

Rules for overriding

(1)In Overriding, Method names and argument types must be matched that is method signatures must be same.

(2)In overriding return types must be same, but this rule is applicable until 1.4 version only.

From 1.5 version onwards, we can take covariant return types. According to this, Child class method return type need not to be same as parent method return type, its child type also allowed.

public class P {

public void property () {

System.out.println ("cash+loan+gold");

}

Public Object marry () {

System.out.println ("subbalaxmi");

}

}

Public class C extends P {

Public String marry () {

System.out.println ("Trisha or charmi");

}

}

Note: Above code is invalid upto java 1.4 version.

Parent Class Method Return Type: Child Class Method return Type: valid/Not Valid

Object Object|String|StringBuffer Valid (P->C)

Number Number|Integer Valid (P->C)

String Object Not Valid(C -> P)

Double int NV (NV for primitive type)

Covariant return type concept applicable only for object types but not for primitive types.

(3) Access Modifiers:

Parent class private methods not available to child class and hence overriding concept not applicable for private methods.

Private:

Class P {

Private void m1 () {

}

}

Class C extends P {

Private void m1 () {

}

}

Note: Above Example will not throw any compile time issue, But Not Overriding.

Based on our requirement we can define exactly same private method in child class, it is valid but not overriding

We cannot override parent class final methods in child classes, if we are trying to override we will get compile time error.

Class P {

Public final void m1 () {

}

}

Class C extends P {

Public final void m1 () {

}

}

Compile time error: m1 () in c cannot override m1 in p, Overridden method is final.

(5) Parent class abstract methods we can override in child class to provide implementation

Abstract class p {

Public abstract void m1 () {

}

}

Class C extends P {

Public void m1() {

}

}

(6)We can override non-abstract methods as abstract

Class P {

Public void m1 () {

}

}

Abstract Class c extends P {

Public abstract void m1 ();

}

Note: Main advantage of above approach is, we can stop the availability of parent method implementation to the next level child classes.

In overriding, the following modifiers will not keep any restriction

Synchronized, native, strictfp:

Parent Class Method Child Class Method: Valid/Invalid

Final nonfinal/final Not Valid

Non-final final Valid

Abstract/Non-abstract Non-Abstract/Abstract Valid

Synchronized/Non-Synchronized Non-Synchronized/Synchronized Valid

Native/non-native non-native/native Valid

Strictfp/non-strictfp non-strictfp/strictfp Valid

(7)While overriding we cannot reduce scope of access modifier but we can increase the scope.

Class P {

Public void m1 () {

}

}

Class C extends P {

void m1() {

}

}

CE: m1 in c cannot override m1 () in p; Attempting to assign weaker access privileges, was public

Access Modifier order: private < default < protected < public

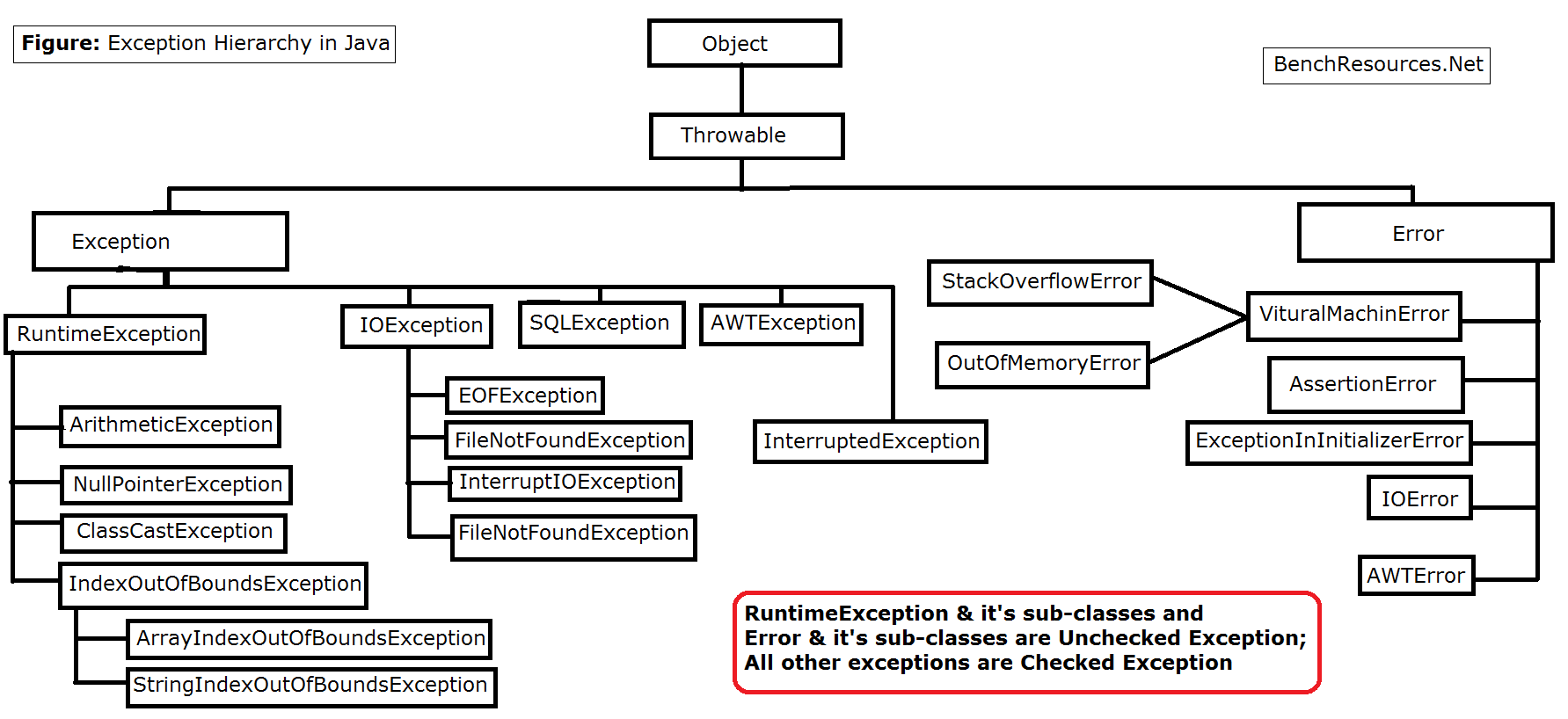
Parent class method: Child class method

Public Public

Protected Public | protected

Default default | protected | public

Private Overriding concept not applicable



Exception class overriding rules:

If Child class method throws any checked exception, then compulsory parent class method should throw the same checked exception or its parent, otherwise we will get compile time error. However, there is no restrictions on unchecked exception.

Example:

Class P {

Public void m1() throws IOException {

}

}

Class C Extends P {

Public void m1 () throws EOFException, Interrupted Exception {

}

}

CE: m1 () in c cannot override m1 () in P; overridden method does not throw java.lang.interruptedException

Scenario Valid/Invalid

1. P: public void m1() throws Exception Valid

C: public void m1 ()

1. P:public void m1()

C: public void m1 () throws Exception Not Valid

(3) P: public void m1 () throws Exception

C: public void m1 () throws Io Exception Valid

(4) P: public void m1 () throws Io Exception Invalid

C: public void m1 () throws Exception

(5) P: public void m1 () throws Io Exception Valid

C: public void m1 () throws FileNotFoundException, EoF Exception

(6) P: public void m1 () throws Io Exception Invalid

C: public void m1 () throws EoF Exception, Interrupted Exception

(7) P: public void m1 () throws Io Exception Valid

C: public void m1 () throws AE, NPE, CCE

Overriding With respect to static methods:

We cannot override a static method as non-static; otherwise, we will get compile time error.

Example:

Class P {

public static void m1 () {

}

}

Class C extends P {

public void m1() {

}

}

CE: m1 () in c cannot override m1 () in overridden method is static.

Similarly, we cannot override a non-static method as static

Example:

Class P {

public void m1 () {

}

}

Class C extends P {

Public static void m1 () {

}

}

If both parent and child class methods are static then we won’t get any compile time error, It seems Overriding concept is applicable for static methods but it is not overriding and it is method hiding

Example:

Class P {

public static void m1 () {

}

}

Class C extends P {

Public static void m1 () {

}

}

Method Hiding:

All rules of method hiding are exactly same as overriding except the following differences.

Diff between Method Hiding and overriding

Method Hiding Overriding

Both parent and child classes should be static both parent and child classes should be non-static

Complier is responsible for method resolution Jvm is always responsible for method resolution

Based on ref type.

Compile time polymorphism or static polymorphism Runtime polymorphism or dynamic binding

Early binding late binding

Programme:

Class P {

Public static void m1 () {

System.out.print (“Parent”);

}

}

Class C extends P {

Public static void m1 () {

System.out.print (“Child”);

}

}

Note: Above is method hiding but is not overriding

Class Test {

Public static void main (String [] args) {

P p = new P ();

p.m1 (); -> parent Method.

C c = new C ();

c.m1 (); -> child Method.

P parentOne = new C ();

parentOne.m1 (); -> Parent

}

}

If both parent and child class methods are non-static then it will become overriding. IN this case, output is parent, child, and child.

Overriding with respect to vararg methods:

We can override vararg method with another vararg method only, if we are trying to override with normal method then it will become overloading but not overriding.

Class p {

Public static void m1 (int ….x) {

Sysout (“Parent”);

}

}

Class c extends p {

Public static void m1 (int x) {

Sysout (“Child”);

}

}

Class Test {

Public static void main (String [] args) {

P p = new P ();

p.m1 (10); -> Parent

C c = new C ();

c.m1 (10); -> Child

P p1 = new C ();

P1.m1 (10); -> parent

}

}

Note: Above is overloading ant not overriding

In the above programme, if we replace child method with vararg method then it will become overriding.

In this case output is

Parent, child, child

Overriding with respect to variable:

Variable resolution always takes care by compiler based on reference type irrespective of whether the variable is static or non-static. (Overriding concept applicable only for methods but not variables.)

class P {

Int x = 888;

}

class C extends P {

Int x = 999;

}

Class Test {

Public static void main (String [] args) {

P p = new p ();

Sysout (p.x); -> 888

C c = new C ();

Sysout (c.x); -> 888

P p1 = new C ();

Sysout (p1.x); -> 888

}

}

P Type C Type Result

P -> non-static p -> non-static 888,999,888

C -> static c -> non-static 888,999,888

P -> non-static p -> static 888,999,888

C -> static c -> static 888,999,888

Differences Between overloading and overriding

Property Overloading Overriding

Method Names Must be same must be same

Argument Types Must be different must be same (Including Order)

(At least order)

Method Signature Must be different must be same

Return Types No Restrictions must be same until 1.4

From 1.5v covariant, return types allowed.

Private, static, final methods can be overloaded cannot be overridden

Access modifier No resection cannot be reduces, but we can increase

Throws clause No Restriction If Child class methods throws any

Checked exception; compulsory

Parent class method should throw the

Same checked exception or its parent

But no restrictions for unchecked exceptions

Method Resolution compiler, based on ref type jvm based on run time object

It is also known compile time static pol|early binding runtime poly|dynamic poly|late

Binding

Note:

In overloading we have to check only method, names (Must be same) and argument types (Must be

Different)

We are not required to check remaining like return types, access modifier etc.

However, in overriding, everything we have to check as if method names, argument types, return types,

Access modifier, throws clause

Consider the following method in parent class

Questions:

Public void m1 (int x) throws Io Exception

In the child, class which of the following methods are valid?

Public void m1 (int x) throws IoExcepton ----> Overriding----->Valid

Public static int m1 (long l) -> Overloading----->Valid

Public static void m1 (int i) ---> overriding -------> Not valid

Public void m1 (int i) throws Exception ----->Not valid (Not a concept of overriding)

Public static abstract void m1 (double d) CE: Illegal compilation of modifier

Polymorphism:

One name but multiple forms is the concept of polymorphism.

Example 1:

Method name is the same but we can apply for different types of arguments (Overloading)

Example: abs (int) abs (long)

Example 2: Method signature is same but in parent class one kind of implementation and in child class

Another type of implementation (Method overriding)

Class P {

marry () {

System.out.println (“subbalaxmi”);

}

}

Class C extends P {

marry () {

System.out.println (“3sha | charmi”);

}

}

Example 3:

Usage of parent reference to hold child object is the concept of polymorphism.

List l = new Array List ();

= new Linked List ();

= new Stack ();

= new Vector ();

Parent class reference can be used to hold child object, but by using that reference we can call only

The methods available in parent class and we cannot call child specific methods.

P -> m1 ();

C -> m2 ();

P p = new C ();

p.m1 ();

p.m2 () ;-----> not valid CE: cannot find symbol

Symbol: method m2 ()

Location: class P

But By using child reference we can call both parent and child class methods

C c = new C ();

c.m1 ();

c.m2 ();

When we should go for Parent reference to hold child object?

If we don’t know exact runtime type of object then we should go for parent reference

For example, the first element present in array list can be any type it may be student or customer object

Alternatively, string or string buffer object hence the return type of get method is object, which can

Hold any object.

Object o = l.get (0);

C c = new C () P p = new C ();

Example Al l = new Al (); List l = new AL ();

We can use this approach if we know exact runtime type of obj we can use this approach if we do not know exact run time type of object

By using this approach, we can call child class and parent class methods by using parent ref we can call only methods available in parent class and we cannot call

Child specific methods. (Dis adv of this approach)

We can use child reference to hold only particular child class obj can use parent ref to hold

(Dis advantage) any child class obj. (advantage)

Note: Encapsulation polymorphism inheritance are three pillars of oops.

PolyMorphism

Compile time polymorphism Dynamic polymorphism

Example: Example : Overrding

Overloading

Method Hiding