# Select control

## switch-case

### common patterns

* The decision variable in switch statement must be variable or expression in char/byte/short/int, Charater/Byte/Short/Integer, Enum, String.
* The case statement must be constants with the same type. The constants could be value of char/byte/short/int/long/String, or final variable, or constant expression. but the constant value of each case statement should be unique. Barring any VM limits, there is no limit to the number of case statements.
* default statement is at most one or omitted. It is usually defined and placed at the end of case statements. but default statement could anywhere in the switch statement including the beginning, middle, or end.
* logic: select entrance that matches value, and execute all statements till meeting break statement or the end of switch block.

|  |
| --- |
| {} of switch block can't be omitted. those of case/default statements could omitted. |
| static void test1(int day){  switch(day){  case 'M':{  System.out.print("Monday ");  break;  }  case 'T':{  System.out.print("Tuesday ");  break;  }  default:{  System.out.print("Nothing ");  break;  }  }  } |
| static void test1(int day){  switch(day){  case 'M':  System.out.print("Monday "); break;  case 'T':  System.out.print("Tuesday "); break;  default:  System.out.print("Nothing "); break;  }  } |

Apply switch-case into selection of objects

|  |
| --- |
| output:  object A  object B  Exception in thread "main" java.lang.IllegalArgumentException: wrong object  at FlowSwitchObj.test(FlowSwitchObj.java:9)  at FlowSwitchObj.main(FlowSwitchObj.java:16) |
| public class FlowSwitchObj {  static void test(Object obj) throws Exception{  //get class name in String type  **switch(obj.getClass().getSimpleName()){**  case "A":  System.out.print("object A"); break;  case "B":  System.out.print("object B"); break;  default:  throw new IllegalArgumentException("wrong object");  }  System.out.print("\n");  }  public static void main(String[] args) throws Exception {  test(new A());  test(new B());  test(new StringBuilder("ABC"));  }    }  class A{}  class B{} |

|  |
| --- |
| can't use Boolean type directly. |
| public class SwitchBoolean {  static void test(boolean val){  String s = String.valueOf(val);  switch(s){  case "true":  System.out.println("true");break;  case "false":  System.out.println("false");break;  default:  System.out.println("nothing");break;  }  }  public static void main(String[] args) {  test(true);  test(false);  }  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### selection

* If some break statements are omitted, one entrance would be selected and then the left statement till end would be executed.
* It is suggested that break statement might be placed at the end of each case/default statement though that is not required.
* The case and default statements could be disorder though the code would unreadable.
* case statement could be empty though that is not suggested.
* {} of switch block can't be omitted. but {} of case/default statements could omitted.

|  |
| --- |
| if day="M", output is Monday Tuesday Nothing. |
| static void test2(int day){  switch(day){  case 'M':  System.out.print("Monday ");  case 'T':  System.out.print("Tuesday ");  default:  System.out.print("Nothing ");  }  System.out.print("\n");  } |

|  |
| --- |
| test3('M'); -> Monday  test3('T'); -> Tuesday  test3('N'); -> Nothing |
| static void test3(int day){  switch(day){  default:  System.out.print("Nothing ");break;  case 'M':  System.out.print("Monday "); break;  case 'T':  System.out.print("Tuesday ");break;  }  System.out.print("\n");  } |

|  |
| --- |
| test5('M');->Monday  test5('T');->Nothing  test5('N');->Nothing |
| static void test5(int day){  // must use final if d is used in case statement  final int d = 'T';  switch(day){  case 'M':  System.out.print("Monday "); break;  **case d:**  default:  System.out.print("Nothing ");break;  } |

|  |
| --- |
| output:  Tuesday  Nothing |
| static void test(int day){  **switch(day+1){**  case 1:  System.out.print("Monday "); break;  case 2:  System.out.print("Tuesday "); break;  default:  System.out.print("Nothing "); break;  }  System.out.print("\n");  }  public static void main(String[] args) throws Exception {  test(1);  test('T');  } |
| output: Monday |
| static void test(int day){  **switch(day/2){**  case 1:  System.out.print("Monday "); break;  case 2:  System.out.print("Tuesday "); break;  default:  System.out.print("Nothing "); break;  }  System.out.print("\n");  }  public static void main(String[] args) throws Exception {  test(2);  } |

|  |
| --- |
| output: 01 |
| public class SwitchEntrance {  static void test1(int day){  switch(day){  **default: System.out.print(0);**  **case 1:**  **System.out.print(1); break;**  case 2:  System.out.print(2); break;  }  System.out.print("\n");  }  public static void main(String[] args){  test1(3);  }  } |

|  |
| --- |
| output: 12  case statement could be final variable. |
| public class SwitchFinal {  static void test1(int day){  final int a=1, b =2;  switch(day){  case a:  System.out.print(1); break;  case b:  System.out.print(2); break;  }  System.out.print("\n");  }  public static void main(String[] args){  test1(1);  test1(2);  }  } |

|  |
| --- |
| output: Good  Those bold statements in one line are little weird statements but the compiling is good. |
| String year = "Senior";  switch(year) {  **case "Freshman" : default: case "Sophomore" :**  case "Junior" : System.out.print("See you"); break;  case "Senior" : System.out.print("Good");  } |

|  |
| --- |
| output: 4  Note: the below show all possible constant in case statements |
| final int movieRating = 4;  int badMovie = 0;  switch(badMovie) {  case 0:  case movieRating: System.out.println("4"); break;  default:  case (int)'a':  case 'b':  case 1\*1: System.out.println("1"); break;  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### enum

|  |
| --- |
| output: 1 |
| public class SwitchEnum {  enum Day{  Mon, Tue  }  static void test1(Day day){  switch(day){  case Mon:  System.out.print(1); break;  case Tue:  System.out.print(2); break;  }  System.out.print("\n");  }  public static void main(String[] args) {  test1(Day.Mon);  }  } |

|  |
| --- |
| output: class A |
| public class SwitchEnum2 {  enum O{ A, B}  static void test1(O d){  switch(d){  case A:  new A(); break;  case B:  new B(); break;  }  System.out.print("\n");  }  public static void main(String[] args) {  test1(O.A);  }  }  class A{  A(){  System.out.println("class A");  }  }  class B{  B(){  System.out.println("class B");  }  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## If-else

|  |
| --- |
| output: bd |
| boolean a = false;  if(a) System.out.print("a");  // a=true return true  **if(a=true) System.out.print("b");**  if(a==false) System.out.print("c");  if(a) {  System.out.print("d");  } |

|  |
| --- |
| output: BearShark  tricky problem due improper indent typing |
| public class BearOrShark {  public static void main(String[] args) {  int luck = 10;  if((luck˃10 ? luck++: --luck)˂10) {  System.out.print("Bear");  } if(luck˂10) System.out.print("Shark");  }  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## break and continue

A break statement transfers flow of control out of enclosing statement.

|  |
| --- |
|  |
|  |

A continue statement within a loop transfers flow of control to the loop continuation point of the loop.

# Iteration

The collection in a for loop could be an array or other objects that are defined in java.lang.Iterable.

## standard approach

for-loop, while-loop, do-while-loop

Scope of loops are encompassed by curly brackets {}. If {} is omitted, the scope only include the next single statement.

|  |
| --- |
|  |
| for(int i=0;i<10;i++){  ….  } |
| for(int i=0;i<10;i++)  ….//one statement |

|  |
| --- |
| int i =0;  while(i<10){  ….  i++;  } |
| int i =0;  while(i<10)  i++; |

|  |
| --- |
|  |
| int i =0;  do{  ….  i++;  } while(i<10); |
|  |

## loop of array

the three approaches are equal. The 2nd is suggested.

|  |
| --- |
|  |
| static void test1(){  int a[] = {1,2,3};  **for(int i = 0; i < a.length; i++){**  System.out.print(a[i]);  }  System.out.println("");  } |
| static void test2(){  int a[] = {4,5,6};  **for(int i : a) //must declare local variable in for-loop**  System.out.print(i);  System.out.println("");  } |
| static void test3(){  int i=0, a[] = {1,2,3};  **for(;;){//infinite loop**  System.out.print(a[i++]);  if (i>=a.length) break;  }  System.out.println("");  } |

The approaches below are equal

|  |
| --- |
|  |
| static void test1(){  int i=0, a[] = {1,2,3};  while(i<a.length){  System.out.print(a[i]);  i++;  }  System.out.println("");  } |
| static void test3(){  int i=0, a[] = {1,2,3};  while(i<a.length)  System.out.print(a[i++]);  System.out.println("");  } |
| static void test4(){  int i=-1, a[] = {1,2,3};  while(i++<a.length-1)  System.out.print(a[i]);  System.out.println("");  } |

|  |
| --- |
|  |
| static void test2(){  int i=0, a[] = {1,2,3};  do{  System.out.print(a[i]);  i++;  }while(i<a.length);//don't forget ;  System.out.println("") ;  } |

## complex loop

### simplified loop

some loop is a little tricky, which is hard to read.

Note: semicolon ";" must be two can't be omitted.

|  |
| --- |
| infinite loop |
| static void test1(){  **for(;;)**  break;  } |

|  |
| --- |
| output: 123 |
| static void test2(){  int a[]= {1,2,3};  **for(int i=-1; ++i<a.length;)**  System.out.print(a[i]);  } |
| static void test3(){  **int $[]= {1,2,3};**  **for(int \_$A=-1; ++\_$A<$.length;)**  **System.out.print($[\_$A]);**  } |

|  |
| --- |
| output:123 |
| static void test4(){  int i = 1;  while(i<4)  System.out.print(i++);  } |
| static void test5(){  int i = 0;  while(i<3)  System.out.print(++i);  } |

|  |
| --- |
| output: 123 |
| static void test6(){  int i = 0;  do{  System.out.print(++i);  }while(i<3);  } |

|  |
| --- |
| output: cdef |
| public class PrintCharacters {  public static void main(String[] args) {  char value = 'c';  **do System.out.print(value++);**  **while (value ˂= 'f');**  }  } |

|  |
| --- |
| output:4  the print statement is actually out of the nested for loop though the indent line may make you confused. |
| int[] times [] = new int[3][3];  for (int i = 0; i < times.length; i++)  for (int j = 0; j < times.length; j++)  times[i][j] = i\*j;  **System.out.println(times[2][2]);** |

|  |
| --- |
| output:  -1 0 1 2 3 4 5 6 |
| public class Test{  public static void main(String[] args) {  int y = -2;  **do System.out.print(++y + " ");**  **while(y <= 5);**  }  } |

|  |
| --- |
|  |
|  |

### nested loop

Keep being watchful when nested loop is used.

|  |
| --- |
| abcdefghi |
| static void test1(){  char c = 'a';  for(int i=1; i<= 3; i++)  for(int j=0; j<= 2; j++) System.out.print(c++);  } |

|  |
| --- |
| output: 2 |
| public class ForNested2 {  public static void main(String[] args) {  int count = 0;  ROW\_LOOP: for(int row = 1; row <=3; row++)  for(int col = 1; col <=2 ; col++) {  if(row \* col % 2 == 0) continue ROW\_LOOP;  count++;  }  System.out.println(count);  }  } |

|  |
| --- |
|  |
|  |

### 2-D array

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### two arrays

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### complex conditions

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

# Exception

## exception statement

employ try-catch or throw to handle exceptions.

No exception is considered.

|  |
| --- |
| output: Exception in thread "main" java.lang.ArithmeticException: / by zero  at ExceptionTest.main(ExceptionTest.java:3) |
| public class ExceptionTest {  public static void main(String[] args) {  System.out.println(5/0);  }  } |

|  |
| --- |
| approach without exception. It is ok but not suggested. |
| public class ExceptionTest {  public static void main(String[] args) {  int a=5, b=0;  if(b==0)  System.out.println("wrong");  else  System.out.println(a/b);  }  } |

In practice, three approaches for handling exceptions are available.

handle exception using "throw"

|  |
| --- |
| output: Exception in thread "main" java.lang.ArithmeticException: dominator is 0  at ExceptionTest.main(ExceptionTest.java:5) |
| public class ExceptionTest {  **public static void main(String[] args) throws Exception {**  int a=5, b=0;  if(b==0)  **throw new ArithmeticException("dominator is 0");**  else  System.out.println(a/b);  }  } |

use try-catch to handle exceptions

|  |
| --- |
| cause exception: java.lang.ArithmeticException: / by zero |
| public class ExceptionTest {  public static void main(String[] args){  int a=5, b=0;  try{  System.out.println(a/b);  }catch(ArithmeticException e){  System.out.println(e);  }  }  } |

use customary exceptions

|  |
| --- |
| define customary exception class which extends java.lang.exceptions |
| public class ExceptionCustomary{  public static void main(String[] args) throws MyExceptions{  int a=5, b=0;  if(b==0){  throw new MyExceptions("zero");  }  else{  System.out.println(a/b);  }  }  }  class MyExceptions extends Exception{  MyExceptions(String msg){  super(msg);  }  } |

|  |
| --- |
|  |
|  |

### try-catch exceptions

Here is standard exception statements

try{

    ...

}catch(Exception e){

    ...

}finally{

    ...

}

* The try-, catch-, or finally- block should be encompassed by curly brackets {}.
* Order of blocks matters: Only one of the patterns of "try-catch", "try-finally", or "try-catch-finally" is allowed.
* Catch blocks could be none or one or multiple. The order of exceptions in catch blocks matters: first sub-exceptions then broader exceptions.
* Finally block could be none or at most one. If "throw exceptions" are defined in try, catch and finally blocks. Only that in finally block will be thrown.

|  |
| --- |
|  |
| public class ExceptionTryCatch {  static void test(int a){  try{  System.out.println(10/a);  }catch(ArithmeticException e){  System.out.println(e);  }catch(Exception e){  System.out.println(e);  }finally{  System.out.println("final");  }  System.out.println("test");  }  public static void main(String[] args){  test(0);  System.out.println("done");  }    } |
| try{  System.out.println(10/a);  }catch(Exception e){  System.out.println(e);  } |
| try{  System.out.println(10/a);  }finally{  System.out.println("final");  } |

### throw exceptions

Two steps: declare exceptions, and then throw exceptions

Classes listed in the throws part of a method declaration must extend java.lang.Throwable. This includes Error, Exception, and RuntimeException.

|  |
| --- |
| declare exceptions firstly and then throw exception in method body |
| public class ExceptionThrows {  public static void main(String[] args) **throws Exception** {  **throw new Exception();**  }  } |
| public class ExceptionThrows {  public static void main(String[] args) **throws Exception** {  **throw new Exception("throw exception");**  }  } |
| public class ExceptionThrows {  public static void main(String[] args) **throws FileNotFoundException** {  **throw new FileNotFoundException();**  }  } |

|  |
| --- |
| declaration exception is not required when handle runtime exception. but including declaration is ok, too. |
| public class ExceptionThrows {  public static void main(String[] args){  **throw new RuntimeException();**  }  } |
| public class ExceptionThrows {  public static void main(String[] args) **throws Exception** {  **throw new RuntimeException();**  }  } |

|  |
| --- |
| constructor throw exceptions |
| public class ConstructorTest{  String name;  **ConstructorTest(String name) throws IllegalArgumentException{**  if(name != null){  this.name = name;  }else{  throw new IllegalArgumentException("null");  }  }  public static void main(String[] args) {  String a = null;  ConstructorTest c = new ConstructorTest(a);  System.out.println(c.name);  }  } |

|  |
| --- |
|  |
|  |

## types of exceptions

### checked exceptions

Checked exceptions should be declared before handling.  
**java.lang.Exceptions** or its subclass is thrown by programmer's code (the package java.lang). The class java.lang.Exception is default import

**IOExceptions** and **FileNotFoundExceptions** ared defined by package "java.io". They are not default import. Should import the class explicitly by "import java.io.IOException;" or "import java.io.FileNotFoundException;" before declaration.

### unchecked exceptions

**java.lang.RuntimeExceptions** is thrown by JVM or programmer's code. It is to call runtime exceptions without declaration but declaration in advance will not hurt. **IllegalArgumentException and NumberFormatException are usually thrown by programmer**

ArithmeticException

ClassCastException

ArrayIndexOutOfBoundsException

IllegalArgumentException

NumberFormatException

### java.lang.Error

A program must handle or declare checked exceptions. It should not handle or declare java.lang.Error.

IOException

Exception

RuntimeException

FileNotfoundException

NullPointerException

ClassCastException

ArithmeticException

ArrayIndexOutOfBoundsException

IllegalArgumentException

NumberFormatException

|  |  |
| --- | --- |
| exceptions | examples |
| NullPointerException | String c = null;  c.length(); |
| ArithmeticException | int a=3, b=0;  a/b; |
| ClassCastException | String a= "abc";  Integer b = (Integer) a; |
| Object obj = new Integer(3);  String s = (String) obj; |
| int[] nums = new int[] { 1, 4, 6};  Object p = nums;  String[] two = (String[]) p; |
| ArrayIndexOutOfBoundsException | int[] a= new int[0];  a[0]; |
|  | int[] a= new int[10];  a[-1]; |
| NumberormatException  IllegalArgumentException | int x = Integer.parseInt("abc") |

## exception chaining

* If the try block, catch block, and finally block all throw an exception, only the final block gets thrown to the caller.
* Exception may not be before any other checked exception.
* rear exception should be wider than previous exceptions. RuntimeException should follow other subclass of RuntimeException.

### flow-control

|  |
| --- |
| output: 12456 |
| public class ExceptionThrow2 {  public String name;  public void run() {  **System.out.print("1");**  try {  **System.out.print("2");**  name.toString();  System.out.print("3");  }catch (NullPointerException e) {  **System.out.print("4");**  }  **System.out.print("5");**  }  public static void main(String[] args) {  ExceptionThrow2 jerry = new ExceptionThrow2();  jerry.run();  **System.out.print("6");**  }  } |

|  |
| --- |
| output:  124Exception in thread "main" 4java.lang.NullPointerException: Cannot invoke "String.toString()" because "this.name" is null          at ExceptionThrow2.run(ExceptionThrow2.java:7)          at ExceptionThrow2.main(ExceptionThrow2.java:17) |
| public class ExceptionThrow2 {  public String name;  public void run() {  **System.out.print("1");**  try {  **System.out.print("2");**  **name.toString(); // The stack trace for a NullPointerException**  System.out.print("3");  }catch (NullPointerException e) {  **System.out.print("4");**  throw e;  }  System.out.print("5");  }  public static void main(String[] args) {  ExceptionThrow2 jerry = new ExceptionThrow2();  jerry.run();  System.out.print("6");  }  } |
| public class ExceptionThrow2 {  public String name;  public void run() {  System.out.print("1");  try {  **System.out.print("2");**  **name.toString(); // The stack trace for a NullPointerException**  System.out.print("3");  }catch (NullPointerException e) {  **System.out.print("4");**  **throw new RuntimeException();**  }  System.out.print("5");  }  public static void main(String[] args) {  ExceptionThrow2 jerry = new ExceptionThrow2();  jerry.run();  System.out.print("6");  }  } |

|  |
| --- |
| 12Exception in thread "main" 45java.lang.NullPointerException: Cannot invoke "String.toString()" because "this.name" is null          at ExceptionThrow2.run(ExceptionThrow2.java:7)          at ExceptionThrow2.main(ExceptionThrow2.java:18) |
| public class ExceptionThrow2 {  public String name;  public void run() {  System.out.print("1");  try {  **System.out.print("2");**  **name.toString();**  System.out.print("3");  }catch (Exception e) {  **System.out.print("4");**  **throw e;**  }finally{  **System.out.print("5");**  }  }  public static void main(String[] args) {  ExceptionThrow2 jerry = new ExceptionThrow2();  jerry.run();  System.out.print("6");  }  } |

|  |
| --- |
| 12Exception in thread "main" 5java.lang.NullPointerException: Cannot invoke "String.toString()" because "this.name" is null          at ExceptionThrow2.run(ExceptionThrow2.java:7)          at ExceptionThrow2.main(ExceptionThrow2.java:18) |
| public class ExceptionThrow2 {  public String name;  public void run() {  **System.out.print("1");**  try {  **System.out.print("2");**  **name.toString();**  System.out.print("3");  }catch (ArithmeticException e) {  System.out.print("4");  throw e;  }finally{  **System.out.print("5");**  }  }  public static void main(String[] args) {  ExceptionThrow2 jerry = new ExceptionThrow2();  jerry.run();  System.out.print("6");  }  } |

|  |
| --- |
| 125Exception in thread "main" java.lang.Exception          at ExceptionThrow2.run(ExceptionThrow2.java:14)          at ExceptionThrow2.main(ExceptionThrow2.java:19) |
| public class ExceptionThrow2 {  public String name;  public void run() throws Exception {  System.out.print("1");  try {  System.out.print("2");  name.toString();  System.out.print("3");  }catch (ArithmeticException e) {  System.out.print("4");  throw new Exception();  }finally{  System.out.print("5");  throw new Exception();  }  }  public static void main(String[] args) throws Exception {  ExceptionThrow2 jerry = new ExceptionThrow2();  jerry.run();  System.out.print("6");  }  } |

|  |
| --- |
| output: 1245 |
| public class Dog {  public String name;  public void parseName() {  **System.out.print("1");**  try {  **System.out.print("2");**  **int x = Integer.parseInt(name);**  System.out.print("3");  } catch (NumberFormatException e) {  **System.out.print("4");**  }  }  public static void main(String[] args) {  Dog leroy = new Dog();  leroy.name = "Leroy";  leroy.parseName();  **System.out.print("5");**  } } |

### multiple catch blocks

|  |
| --- |
| output:  a  b  c  e  Exception in thread "main" java.lang.RuntimeException: 3  at ExceptionChain.main(ExceptionChain.java:15) |
| public class ExceptionChain {  public static void main(String[] args) throws Exception {  System.out.println("a");  try{  System.out.println("b");  throw new IllegalArgumentException();  }catch(IllegalArgumentException e){  System.out.println("c");  throw new RuntimeException("1");  }catch(RuntimeException e){  System.out.println("d");  throw new RuntimeException("2");  }finally{  System.out.println("e");  throw new RuntimeException("3");  }  }    } |

exception masks and nested try-catch blocks

exception defined in catch block will be masked if finally block define another exception, too. Therefore, nested exceptions are needed.

|  |
| --- |
| not compiling: exception mask |
| public class ExceptionMask {  public static void main(String[] args){  try{  throw new RuntimeException();  }catch(RuntimeException e){  throw new RuntimeException();  }finally{  throw new Exception();  }  }  } |

|  |
| --- |
| output: -c |
| public class ExceptionChain2 {  static String s = "-";  public static void main(String[] args){  new ExceptionChain2().go1();  System.out.println(s);  }  void go1(){  try{ go2(); }  catch(Exception e){ s += "c"; }  }  void go2() throws Exception{  **go3(); s += "2"; //Not suggested though compiling is ok. They are unreachable code.**  **go3(); s += "2b";**  }  void go3() throws Exception{  throw new Exception();  }  } |
| compiling of the second class will fail if directly throws exception instead of go3(). |
| public class ExceptionChain2 {  static String s = "-";  public static void main(String[] args){  new ExceptionChain2().go1();  System.out.println(s);  }  void go1(){  try{ go2(); }  catch(Exception e){ s += "c"; }  }  void go2() throws Exception{  throw new Exception(); s += "2";  }  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### nested exceptions

|  |
| --- |
| output: 1345 |
| public class ExceptionMask {  public static void main(String[] args){  String a = null;  try{  try{  System.out.println("1");  a.length();  System.out.println("2");  }catch(NullPointerException e){  System.out.println("3");  throw new RuntimeException();  }finally{  System.out.println("4");  }  }catch(Exception e){  System.out.println("5");  }  }    } |
| //output is 12345. the snippet would not be compiled if uncomment the red statement.  public class ExceptionTest {  void test(){  System.out.println("1");  try{  System.out.println("2");  throw new RuntimeException();  }catch(RuntimeException e){  System.out.println("3");  // throw new RuntimeException();  }finally{  System.out.println("4");  }  System.out.println("5");  }  public static void main(String[] args){  ExceptionTest c = new ExceptionTest();  c.test();  }  } |
| //output: 1234 and exception  public class ExceptionFinally {  void test(){  System.out.println("1");  try{  System.out.println("2");  throw new RuntimeException();  }catch(RuntimeException e){  System.out.println("3");  throw new RuntimeException();  }finally{  System.out.println("4");  throw new RuntimeException();  }  // not compiling if uncomment this  // System.out.println("5");  }  public static void main(String[] args){  ExceptionFinally c = new ExceptionFinally();  c.test();  }  } |

|  |
| --- |
|  |
|  |

## inheritance

When child method invokes parent methods or overrides parent method, the child methods must have the compatible checked exceptions with corresponding methods in parent class. Any mismatches will cause compiling error.

If an interface has the method "void run() throws Exception;", the implementing class should have the method "public void run() throws Exception{ }". The implementing class is allowed to have the method "public void run() { }" OR the method "public void run() throws IOException{ }" OR "public void run() throws RuntimeException{ };".

If an interface has the method "void run() throws RuntimeException;", the implementing class is allowed to have the method "public void run(){ }" OR "public void run() throws RuntimeException{ };" OR "public void run() throws NullPointerException{ };".

If an interface has the method "void run();", the implementing class is **NOT** allowed to have the method "public void run() throws Exception{ }".

If an interface has the method "void run() throws RuntimeException;", the implementing class is **NOT** allowed to have the method "public void run() throws Exception{ }".

### virtual class

|  |
| --- |
| output: parent |
| public class ExceptionInherit3 extends Test{  **public static void main(String[] args) throws Exception{**  new ExceptionInherit3().go();  }  }  class Test{  **void go() throws Exception{**  System.out.println("parent");  }  } |

|  |
| --- |
| output: child |
| public class ExceptionInherit3 extends Test{  **void go() throws Exception{**  System.out.println("child");  }  **public static void main(String[] args) throws Exception{**  new ExceptionInherit3().go();  }  }  class Test{  **void go() throws Exception{**  System.out.println("parent");  }  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### abstract class

|  |
| --- |
| output: concrete |
| **public class ExceptionInherit4 extends Test{**  **void go() throws Exception{**  System.out.println("concrete");  }  public static void main(String[] args) throws Exception{  new ExceptionInherit4().go();  }  }  abstract class Test{  **abstract void go() throws Exception;**  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### interface

|  |
| --- |
| output: concrete |
| **public class ExceptionInterface implements Test{**  **public void go() throws Exception{**  System.out.println("concrete");  }  **public static void main(String[] args) throws Exception{**  new ExceptionInterface().go();  }  }  interface Test{  **void go() throws Exception;**  } |

|  |
| --- |
|  |
|  |

## customary exceptions

* customary class extends the base class Exception
* Throw that customary class at methods.
* override methods with exceptions defined. The exception type should not be omitted and should be consistent

### define and invoke

|  |
| --- |
|  |
| public class ExceptionCustomary{  **public static void main(String[] args) throws MyExceptions{**  int a=5, b=0;  if(b==0)  throw new MyExceptions("zero");  }  }  **class MyExceptions extends Exception{**  MyExceptions(String msg){  super(msg);  }  } |

|  |
| --- |
| In try-catch, the customary exception must be compatible with the exceptions. |
| public class ExceptionCustomary2{  public static void main(String[] args){  int a=5, b=0;  try{  int res = a/b;  **}catch(MyException e){**  System.out.println("wrong");  }  }  }  **class MyException extends ArithmeticException{**  MyException(String msg){  super(msg);  }  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### inheritance

Here, a class extends another class. A customary exception could be added to class method

|  |
| --- |
| public class ExceptionInherit extends Test{  public static void main(String[] args){  new ExceptionInherit().go();  }  void go(){  System.out.println("go");  }  }  class Test{  Test(){  System.out.println("test");  }  } |

|  |
| --- |
| output:  child  parent  Exception in thread "main" myExceptions: child exceptions          at myTest.go(ExceptionInherit.java:22)          at ExceptionInherit.main(ExceptionInherit.java:6) |
| public class ExceptionInherit extends myTest{  ExceptionInherit(){  System.out.println("parent");  }  **public static void main(String[] args) throws myExceptions{**  new ExceptionInherit().go();  }  }  class myExceptions extends Exception{  myExceptions(String msg){  super(msg);  }  }  class myTest{  myTest(){  System.out.println("child");  }  **void go() throws myExceptions{**  throw new myExceptions("child exceptions");  }  } |

**override methods with exceptions defined. The exception type should not be omitted and should be consistent.**

|  |
| --- |
| output:  child  parent  Exception in thread "main" myExceptions: parent exceptions          at ExceptionInherit2.go(ExceptionInherit2.java:6)          at ExceptionInherit2.main(ExceptionInherit2.java:9) |
| public class ExceptionInherit2 extends myTest{  ExceptionInherit2(){  System.out.println("parent");  }  **void go() throws myExceptions{ //myExceptions should exist**  throw new myExceptions("parent exceptions");  }  **public static void main(String[] args) throws myExceptions{//myExceptions should exist**  new ExceptionInherit2().go();  }  }  class myExceptions extends Exception{  myExceptions(String msg){  super(msg);  }  }  class myTest{  myTest(){  System.out.println("child");  }  **void go() throws myExceptions{**  throw new myExceptions("child exceptions");  }  } |

|  |
| --- |
| output:  limp hurt 2 |
| public class ExceptionInherit1 {  private static void split() throws HurtException {  throw new HurtException();  }  static void run(){  try{  split();  }catch(LimpException e){  System.out.print("2 ");  }catch(Exception e){  System.out.print("3 ");  }  }  public static void main(String[] args) throws Exception {  run();  }  }  class LimpException extends Exception{  LimpException(){  System.out.print("limp ");  }  }  class HurtException extends LimpException{  HurtException(){  System.out.print("hurt ");  }  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |