Syllabus

create checked exception

Method or constructor raising the checked exception

caller of that method or constructor has to either handle or declare the checked exception.

public class CheckedException extends Exception {

public CheckedException(String message) {

super(message);

// **TODO** Auto-generated constructor stub

}

}

public class Raiser {

Raiser() throws CheckedException{

throw new CheckedException("Yeah this is a checked exception !");

}

}

public class Demo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

try {

Raiser r = new Raiser();

}

catch(CheckedException e) {

e.printStackTrace();

}

System.***out***.println("Khel Khatam Paisa Hajam ");

}

}

non abstract Parent class or abstract parent class or interface with some methods

child classes of parent class or implementation classes of interface.

a) create array of non abstract class or abstract class or interface. Store child class objects

traverse the array and call methods of child classes (which are available in the parent class/interface)

also call those methods which are not in the parent class ( downcasting )

public interface interfacu {

void hello();

void bye();

}

public class Child1 implements interfacu {

*@Override*

public void hello() {

// **TODO** Auto-generated method stub

System.***out***.println("Hello How are ? mai,"+getClass().getName() + " hu, tum Khana khake jana ha....");

}

*@Override*

public void bye() {

// **TODO** Auto-generated method stub

System.***out***.println("Aee Bye bye");

}

public void child1Method() {

System.***out***.println("child1 se milke accha laga na?? ");

}

}

public class Child2 implements interfacu {

*@Override*

public void hello() {

// **TODO** Auto-generated method stub

System.***out***.println("Hello How are ? mai,"+getClass().getName() + " hu, tum Khana khake jana ha...");

}

*@Override*

public void bye() {

// **TODO** Auto-generated method stub

System.***out***.println("Aee Bye bye");

}

public void child2Method() {

System.***out***.println("child2 se milke accha laga na?? ");

}

}

public class Demo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

interfacu i[] = {new Child1(), new Child2()};

for(interfacu ref : i) {

ref.hello();

if (ref instanceof Child1) ((Child1) ref).child1Method();

if (ref instanceof Child2) ((Child2) ref).child2Method();

ref.bye();

System.***out***.println();

}

}

}

b) some static function with the argument as non abstract parent class or abstract class or interface

call methods of child classes (which are available in the parent class/interface)

also call those methods which are not in the parent class ( downcasting )

[ upcasting, downcasting, instanceof etc. ]

public class Demo {

public static void show(interfacu ref) {

ref.hello();

if (ref instanceof Child1) ((Child1) ref).child1Method();

if (ref instanceof Child2) ((Child2) ref).child2Method();

ref.bye();

}

public static void main(String[] args) {

// **TODO** Auto-generated method stub

interfacu i[] = {new Child1(), new Child2()};

for(interfacu ref : i) {

*show*(ref);

System.***out***.println();

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Write a program to print the first 10 Fibonacci numbers.
2. public class Demo {
3. public static void main(String[] args) {
4. // **TODO** Auto-generated method stub
5. int k=10;
7. int f1=0;
8. int f2=1;
9. int f3=1;
10. System.***out***.println(f1 +"\n" + f2);
11. for(int i =0;i<=k-2;i++) {
12. System.***out***.println(f3);
13. f1=f2;
14. f2=f3;
15. f3 = f1+f2;
16. }
17. }
18. }

2) create a checked exception "VotingNotAllowedException".

define a class "Person" with a instance member "private String name" and "private int age"

define parameterized constructor and toString() method also.

In the parameterized constructor check the age passed while instantiating the class. if the age is less than 18 , constructor should raise "VotingNotAllowedException" [ it should not handle the exception]

Define a class "Demo" with main function.

From main function create an object of "Person" and display that object.

public class VotingNotAllowedException extends Exception {

public VotingNotAllowedException(String messege) {

super(messege);

}

}

public class Person {

private String name;

private int age;

public Person(String name, int age) throws VotingNotAllowedException {

this.name = name;

this.age = age;

if(age < 18) {

throw new VotingNotAllowedException("Voting not allowed");

}

}

public String toString() {

return "name is "+this.name+" and age is "+this.age;

}

}

public class Demo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

try {

Person p = new Person("Abc", 18);

System.***out***.println("Person details : "+p);

}

catch(Exception e) {

e.printStackTrace();

}

}

}

3) Sample s1=new Sample(20);

Sample s2=new Sample (20);

make sure following things:

S.o.p( s1+" " +s2); 20 20

S.o.p(s1.equals(s2)); true

S.o.p(s1.hashCode()) 20

S.o.p(s2.hashCode()) 20

public class Sample {

private int number;

public Sample(int n) {

number = n;

}

public String toString() {

return String.*valueOf*(number);

}

public boolean equals(Object ref) {

Sample temp = (Sample) ref;

if(this.number == temp.number) return true;

return false;

}

public int hashcode() {

return number;

}

public static void main(String[] args) {

Sample s1=new Sample(20);

Sample s2 = new Sample(20);

System.***out***.println(s1+" "+s2);

System.***out***.println(s1.equals(s2));

System.***out***.println(s1.hashcode());

System.***out***.println(s2.hashcode());

}

}

4) accept a character and display (using if.. else ) whether it is vowel or not.

public class Sample {

static void check(char c) {

if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {

System.***out***.println("Vowel");

}

else {

System.***out***.println("not a vowel");

}

}

public static void main(String[] args) {

*check*('a');

}

}

5) public interface PizzaIngredientFactory

{

public Dough createDough();

public Sauce createSauce();

public Cheese createCheese();

public Pepperoni createPepperoni();

public Clams createClams();

}

define 2 implementations

USPizzaIngredientFactory and IndianPizzaIngredientFactory

create an array of "PizzaIngredientFactory" , store all the implemenations object inside it, traverse the array and invoke all the methods.

interface PizzaIngredientFactory

{

public Dough createDough();

public Sauce createSauce();

public Cheese createCheese();

public Pepperoni createPepperoni();

public Clams createClams();

}

class UsPizzaingredientfactory implements PizzaIngredientFactory{

*@Override*

public Dough createDough() {

// **TODO** Auto-generated method stub

return new Dough(getClass().getName());

}

*@Override*

public Sauce createSauce() {

// **TODO** Auto-generated method stub

return new Sauce(getClass().getName());

}

*@Override*

public Cheese createCheese() {

// **TODO** Auto-generated method stub

return new Cheese(getClass().getName());

}

*@Override*

public Pepperoni createPepperoni() {

// **TODO** Auto-generated method stub

return new Pepperoni(getClass().getName());

}

*@Override*

public Clams createClams() {

// **TODO** Auto-generated method stub

return new Clams(getClass().getName());

}

}

class IndPizzaingredientfactory implements PizzaIngredientFactory{

*@Override*

public Dough createDough() {

// **TODO** Auto-generated method stub

return new Dough(getClass().getName());

}

*@Override*

public Sauce createSauce() {

// **TODO** Auto-generated method stub

return new Sauce(getClass().getName());

}

*@Override*

public Cheese createCheese() {

// **TODO** Auto-generated method stub

return new Cheese(getClass().getName());

}

*@Override*

public Pepperoni createPepperoni() {

// **TODO** Auto-generated method stub

return new Pepperoni(getClass().getName());

}

*@Override*

public Clams createClams() {

// **TODO** Auto-generated method stub

return new Clams(getClass().getName());

}

}

public class pizza {

public static void main(String[] args) {

PizzaIngredientFactory[] arr= {new UsPizzaingredientfactory(), new IndPizzaingredientfactory()};

for(PizzaIngredientFactory ref:arr)

{

ref.createDough();

ref.createSauce();

ref.createPepperoni();

ref.createClams();

ref.createCheese();

System.***out***.println();

}

}

}

public class Dough {

Dough(String demo){

System.***out***.println("new Dough created by "+ demo);

}

}

public class Sauce {

Sauce(String demo){

System.***out***.println("sauce added by "+demo);

}

}

public class Pepperoni {

Pepperoni(String demo){

System.***out***.println("Pepperoni added by "+demo);

}

}

public class Clams {

Clams(String demo){

System.***out***.println("clams added by "+demo);

}

}

public class Cheese {

Cheese(String demo){

System.***out***.println("Cheese added by "+demo);

}

}

6) public class Course

{

public void start()

{

}

public void stop()

{

}

}

now define 3 child classes

DACDBDACourse,MSCit and Basic

inside "DACDBDACourse" class define one more method "public void orientation()"

define one more class "CourseDemo" with main function.

inside main function

create an array of "Course" of 3 elements.

store all the child classes objects. Traverse the array and invoke "start()" and "stop()" of all the objects. Also invoke "orientation()" method wherever object of "DACDBDACourse" is present.

public class Course {

public void start()

{

System.***out***.println(getClass().getName()+" Course Started");

}

public void stop()

{

System.***out***.println(getClass().getName()+" Course Ended")

}

}

public class MSCit extends Course {}

public class Basic extends Course {}

public class DACDBDACourse extends Course {

public void orientation() {

System.***out***.println("Orientation of "+getClass().getName());

}

}

public class CourseDemo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

Course c[] = {new Basic(), new MSCit(), new DACDBDACourse()};

for(Course ref : c) {

if(ref instanceof DACDBDACourse) {

((DACDBDACourse) ref).orientation();

}

ref.start();

ref.stop();

}

}

}

7) same as above

define one more class "CourseDemo" with main function and "public static void show()" method.

from main function

invoke "show()" method by passing object of any implementations and invoke "start()" and "stop()" methods. Also invoke "orientation()" method whenever object of "DACDBDACourse" is passed.

public class Course {

public void start()

{

System.***out***.println(getClass().getName()+" Course Started");

}

public void stop()

{

System.***out***.println(getClass().getName()+" Course Ended")

}

}

public class MSCit extends Course {}

public class Basic extends Course {}

public class DACDBDACourse extends Course {

public void orientation() {

System.***out***.println("Orientation of "+getClass().getName());

}

}

public class CourseDemo {

public static void show(Course ref) {

if(ref instanceof DACDBDACourse) {

((DACDBDACourse) ref).orientation();

}

ref.start();

ref.stop();

}

public static void main(String[] args) {

// **TODO** Auto-generated method stub

Course c[] = {new Basic(), new MSCit(), new DACDBDACourse()};

for(Course ref : c) {

*show*(ref);

}

}

}

8) write a class "Displayer" with "print" function

create a class "Sample" with "toString()" method

from main function call "print" by passing following values

"welcome" 100 new Sample

make sure u get following output:

welcome

100

Sample [ class name should be printed hint: getClass().getName()]

public class Displayer {

public void print(String str, int n, Object ref) {

System.***out***.println(str);

System.***out***.println(n);

System.***out***.println(ref);

}

}

public class Sample {

public String toString() {

return getClass().getName();

}

}

public class Demo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

Displayer d = new Displayer();

d.print("welcome", 100, new Sample());

}

}

9) create 2 threads to display "Hello" 5 times each. Use extends Thread technique.

public class Demo extends Thread {

public void run() {

for(int i=0;i<5;i++) {

System.***out***.println("Hello "+ getName() + " "+(i+1)+ " "+ *currentThread*());

}

}

public static void main(String[] args) {

// **TODO** Auto-generated method stub

Demo d1= new Demo();

Demo d2 = new Demo();

d1.setName("d1");

d2.setName("d2");

d1.start();

d2.start();

}

}

10) create 2 threads to display "Welcome" 5 times each. Use implements Runnable technique.

public class Demo implements Runnable {

synchronized public void run() {

for(int i=0;i<5;i++) {

System.***out***.println("Hello "+ Thread.*currentThread*().getName() + " "+(i+1)+ " "+ Thread.*currentThread*());

}

System.***out***.println();

}

public static void main(String[] args) {

// **TODO** Auto-generated method stub

Demo d1= new Demo();

Thread t1 = new Thread(d1);

Thread t2 = new Thread(d1);

t1.setName("t1");

t2.setName("t2");

t1.start();

t2.start();

}

}