Laborator 9 - Exceptii

1. Crearea unei exceptii. Aruncarea exceptiei. Prinderea si tratarea exceptiei

**class** SimpleException **extends** Exception {}

**public** **class** InheritingExceptions {

**public** **void** f() **throws** SimpleException { // metoda este marcata referitor la faptul ca este susceptibila sa arunce o exceptie

System.*out*.println("Throw SimpleException from f()");

**throw** **new** SimpleException(); // se arunca exceptia

}

**public** **static** **void** main(String[] args) {

InheritingExceptions sed = **new** InheritingExceptions();

**try** { // regiune guardata

sed.f();

} **catch**(SimpleException e) { // exceptia este prinsa si rezolvata

System.*err*.println("Caught it!");

}

}

}

1. Creati o clasa de tip exceptie proprie prin utilizarea cuvantului cheie extends. Scrieti un constructor pentru aceasta clasa care sa preia un argument de tip String sis a il salveze in obiectul de tip exceptie ca si o referinta. Scrieti o metoda a clasei de tip exceptie care sa afiseze stringul agregat. Create un cod de tip try-catch in care sa demonstrate functionalitatea create
2. Utilizarea logging

**import** java.util.logging.\*;

**import** java.io.\*;

**class** LoggingException **extends** Exception {

**private** **static** Logger *logger* = Logger.*getLogger*("LoggingException");

**public** LoggingException() {

StringWriter trace = **new** StringWriter();

printStackTrace(**new** PrintWriter(trace));

*logger*.severe(trace.toString());

}

}

**public** **class** LoggingExceptions {

**public** **static** **void** main(String[] args) {

**try** {

**throw** **new** LoggingException();

} **catch**(LoggingException e) {

System.*err*.println("Caught " + e);

}

**try** {

**throw** **new** LoggingException();

} **catch**(LoggingException e) {

System.*err*.println("Caught " + e);

}

}

}

1. Rearuncarea unei exceptii

**public** **class** Rethrowing {

**public** **static** **void** f() **throws** Exception {

System.*out*.println("originating the exception in f()");

**throw** **new** Exception("thrown from f()");

}

**public** **static** **void** g() **throws** Exception {

**try** {

*f*();

} **catch**(Exception e) {

System.*out*.println("Inside g(),e.printStackTrace()");

e.printStackTrace(System.*out*);

**throw** e;

}

}

**public** **static** **void** h() **throws** Exception {

**try** {

*f*();

} **catch**(Exception e) {

System.*out*.println("Inside h(),e.printStackTrace()");

e.printStackTrace(System.*out*);

**throw** (Exception)e.fillInStackTrace();

}

}

**public** **static** **void** main(String[] args) {

**try** {

*g*();

} **catch**(Exception e) {

System.*out*.println("main: printStackTrace()");

e.printStackTrace(System.*out*);

}

**try** {

*h*();

} **catch**(Exception e) {

System.*out*.println("main: printStackTrace()");

e.printStackTrace(System.*out*);

}

}

}

1. Inlantuirea exceptiilor

**class** DynamicFieldsException **extends** Exception {}

**public** **class** DynamicFields {

**private** Object[][] fields;

**public** DynamicFields(**int** initialSize) {

fields = **new** Object[initialSize][2];

**for**(**int** i = 0; i < initialSize; i++)

fields[i] = **new** Object[] { **null**, **null** };

}

**public** String toString() {

StringBuilder result = **new** StringBuilder();

**for**(Object[] obj : fields) {

result.append(obj[0]);

result.append(": ");

result.append(obj[1]);

result.append("\n");

}

**return** result.toString();

}

**private** **int** hasField(String id) {

**for**(**int** i = 0; i < fields.length; i++)

**if**(id.equals(fields[i][0]))

**return** i;

**return** -1;

}

**private** **int** getFieldNumber(String id) **throws** NoSuchFieldException {

**int** fieldNum = hasField(id);

**if**(fieldNum == -1)

**throw** **new** NoSuchFieldException();

**return** fieldNum;

}

**private** **int** makeField(String id) {

**for**(**int** i = 0; i < fields.length; i++)

**if**(fields[i][0] == **null**) {

fields[i][0] = id;

**return** i;

}

// No empty fields. Add one:

Object[][] tmp = **new** Object[fields.length + 1][2];

**for**(**int** i = 0; i < fields.length; i++)

tmp[i] = fields[i];

**for**(**int** i = fields.length; i < tmp.length; i++)

tmp[i] = **new** Object[] { **null**, **null** };

fields = tmp;

// Recursive call with expanded fields:

**return** makeField(id);

}

**public** Object getField(String id) **throws** NoSuchFieldException {

**return** fields[getFieldNumber(id)][1];

}

**public** Object setField(String id, Object value) **throws** DynamicFieldsException {

**if**(value == **null**) {

// Most exceptions don’t have a "cause" constructor.

// In these cases you must use initCause(),

// available in all Throwable subclasses.

DynamicFieldsException dfe = **new** DynamicFieldsException();

dfe.initCause(**new** NullPointerException());

**throw** dfe;

}

**int** fieldNumber = hasField(id);

**if**(fieldNumber == -1)

fieldNumber = makeField(id);

Object result = **null**;

**try** {

result = getField(id); // Get old value

} **catch**(NoSuchFieldException e) {

// Use constructor that takes "cause":

**throw** **new** RuntimeException(e);

}

fields[fieldNumber][1] = value;

**return** result;

}

**public** **static** **void** main(String[] args) {

DynamicFields df = **new** DynamicFields(3);

System.*out*.println(df);

**try** {

df.setField("d", "A value for d");

df.setField("number", 47);

df.setField("number2", 48);

System.*out*.println(df);

df.setField("d", "A new value for d");

df.setField("number3", 11);

System.*out*.println("df: " + df);

System.*out*.println("df.getField(\"d\") : " + df.getField("d"));

Object field = df.setField("d", **null**); // Exception

} **catch**(NoSuchFieldException e) {

e.printStackTrace(System.*out*);

} **catch**(DynamicFieldsException e) {

e.printStackTrace(System.*out*);

}

}

}