Exceptions - What and Why

Concept - Exceptions are events which are not part of an algorithm
Exceptions typically result from errors in input or initialization.
Eg. - An input statement which is expecting an “int” finds a real.
Eg. - The right operand of a divide operator is zero.

Exception handlers are code which express the programmers intentions for dealing with these events.

Exception handlers are needed whenever a program does not have control over its inputs.
Eg. Consider online stores and form content.

Exception handlers are needed whenever a program needs to continue execution even when events which should not occur do occur.
Eg. A program to control an instrument in a biology laboratory.

Java Language Features for Exception Management

Java has extensive features for management of exceptions. We will cover a subset which enables continuing operation in the presence of errors and assists program debugging.

Exception - A class implemented in the java.lang package. An instance of Exception is created to enable documentation of a non-algorithmic event and recovery from the condition induced by the non-algorithmic event.

Language Features

try and catch blocks - Java incorporates a construct for “trying” to execute a step in a program and “catching” and exception if one occurs.

throw - The throw statement creates an instance of the class Exception or some class derived from Exception.

Exceptions in Java

```
public class Account
{  private long accountNumber;
   private long currentBalance;
   public Account( long accountNumber,long initialBalance)
            { this.accountNumber = accountNumber;
              currentBalance = initialBalance;
            }
    public void credit(long amount)
   {        try
            {
            if(amount < 0)
            {
            throw new Exception("Negative credit: "+ amount+ "is not  allowed.");
            }
            setBalance(getBalance()+amount);
            }
            catch (Exception e)
            {System.out.println(e.getMessage());
            }
   }
```


Exceptions in Java

```
public void debit(long amount)
      {try
            {
            if(amount < 0){
            throw new Exception("Negative debit: "+ amount+ "is not allowed.");
      }
      setBalance(getBalance()-amount);
            }
      catch (Exception e)
            {System.out.println(e.getMessage());
            }
   }
   public long getBalance(){
      return currentBalance;
   }
   private void setBalance(long b) {
      currentBalance = b;
   }
}
```

Exceptions in Java

Test Program Output

Test of credit method
current balance = 0
Amount to be credited = 10
New balance is = 10
Test of debit method
current balance = 10
Amount to be debited = 10
New balance is = 0
Test of credit method with negative amount
current balance = 0
Amount to be added = -10
Negative credit: -10 is not allowed.
New balance is =0
Test of debit method with positive debit
current balance = 0
Amount to be debited = -10
Negative debit: -10 is not allowed.
New balance is = 0

Note that since the exception is caught the system continues execution after the exception is thrown.

Exceptions in Java

Let us look at the information on the class Exception which is available on the Java web site.

http://java.sun.com/products/jdk/1.2/docs/api/index.html

java.lang
Class Exception

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
```

Exceptions in Java

public class **Exception**
extends Throwable

The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.

See Also:

`Error`, Serialized Form

Constructor Summary

`Exception()`
Constructs an Exception with no specified detail message.

`Exception(String s)`
Constructs an Exception with the specified detail message.

Exceptions in Java

Methods inherited from class java.lang.Throwable

`fillInStackTrace`, `getLocalizedMessage`, `getMessage`, `printStackTrace`, `printStackTrace`, `printStackTrace`, `toString`

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

We should go and look at Throwable since Exception inherits the methods of Throwable and see what might be useful.

Exceptions in Java

public class Throwable
extends <u>Object</u>
implements <u>Serializable</u>

The `Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or of one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement. Similarly, only this class or one of its subclasses can be the argument type in a catch clause.`

Instances of two subclasses, <u>`Error`</u> and <u>`Exception`</u>, are conventionally used to indicate that exceptional situations have occurred. Typically, these instances are freshly created in the context of the exceptional situation so as to include relevant information (such as stack trace data).

By convention, class `Throwable and its subclasses have two constructors, one that takes no arguments and one that takes a String argument that can be used to produce an error message.`

Exceptions in Java

A `Throwable class contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error.`

Here is one example of catching an exception:

```
            try {
                int a[] = new int[2];
                a[4];
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("exception: " +
       e.getMessage());
                e.printStackTrace();
            }
```

Exceptions in Java

Constructor Summary

Throwable()
Constructs a new Throwable with null as its error message string.

Throwable(String message)

Constructs a new Throwable with the specified error message.

Exceptions in Java

Method Summary

String getMessage()
Returns the errort message string of this throwable object.

void printStackTrace()
Prints this Throwable and its backtrace to the standard error stream.

void printStackTrace(PrintStream s)
Prints this Throwable and its backtrace to the specified print stream.

void printStackTrace(PrintWriter s)
Prints this Throwable and its backtrace to the specified print writer.

String toString()
Returns a short description of this throwable object.

Exceptions in Java

```
public class Account
{  private long accountNumber;
   private long currentBalance;
  public Account( long accountNumber,long initialBalance)
            { this.accountNumber = accountNumber;
              currentBalance = initialBalance;
            }
  public void credit(long amount)
   {        try
            {
            if(amount < 0)
            {
         throw new Exception("Negative credit: "+amount+ "is not allowed.");
      }
            setBalance(getBalance()+amount);
            }
            catch (Exception e)
            {System.out.println(e.getMessage());
             e.printStackTrace(System.out);
            }
   }
```


Exceptions in Java

```
  public void debit(long amount)
        {try
              {
              if(amount < 0){
           throw new Exception("Negative debit: "+amount+ "is not allowed.");
        }
        setBalance(getBalance()-amount);
              }
              catch (Exception e)
              {System.out.println(e.getMessage());
              e.printStackTrace(System.out);
              }
     }
     public long getBalance(){
        return currentBalance;
     }
     private void setBalance(long b) {
        currentBalance = b;
     }
  }
```

Exceptions in Java

Test Program Output

Test of credit method
current balance = 0
Amount to be credited = 10
New balance is = 10
Test of debit method
current balance = 10
Amount to be debited = 10
New balance is = 0
Test of credit method with negative amount
current balance = 0
Amount to be added = -10
Negative credit: -10is not allowed.
java.lang.Exception: Negative credit: -10 is not allowed.
at Account.credit(Account.java:15)
at TestAccount.main(TestAccount.java:25)
New balance is =0

Note Stack Trace is output.

Exceptions in Java

Test of debit method with positive debit
current balance = 0
Amount to be debited = -10
Negative debit: -10 is not allowed.
java.lang.Exception: Negative debit: -10 is not allowed.
at Account.debit(Account.java:28)
at TestAccount.main(TestAccount.java:30)
New balance is = 0

Exceptions in Java

Alternative Mode of Programming with Exceptions

```
public class Account
{  private long accountNumber;
   private long currentBalance;
  public Account( long accountNumber,long initialBalance)
           { this.accountNumber = accountNumber;
             currentBalance = initialBalance;
           }
  /* The credit method interface specifies that it may throw an exception. */
  public void credit(long amount) throws RuntimeException             {
      if(amount < 0){
       throw new RuntimeException("Negative credit: "+amount+ "is not allowed.");
      }
      setBalance(getBalance()+ amount);
   }
   /* Since the Exception is not caught the program will terminate after throwing the
                     exception and printing the stack trace. */
```


Exceptions in Java

```
public void debit(long amount) throws RuntimeException {
      if(amount < 0){
         throw new RuntimeException("Negative debit: "+amount+ "is not allowed.");
      }
      setBalance(getBalance()-amount);
   }
   public long getBalance(){
      return currentBalance;
   }
   private void setBalance(long b) {
      currentBalance = b;
   }
}
```

Exceptions in Java

Direct Known Subclasses: System Defined Exceptions

AclNotFoundException, ActivationException, AlreadyBoundException, ApplicationException, AWTException, BadLocationException, ClassNotFoundException, CloneNotSupportedException, DataFormatException, ExpandVetoException, GeneralSecurityException, IllegalAccessException, InstantiationException, InterruptedException, IntrospectionException, InvocationTargetException, IOException, LastOwnerException, NoninvertibleTransformException, NoSuchFieldException, NoSuchMethodException, NotBoundException, NotOwnerException, ParseException, PrinterException, PrivilegedActionException, PropertyVetoException, RemarshalException, *RuntimeException*, ServerNotActiveException, SQLException, TooManyListenersException, UnsupportedFlavorException, UnsupportedLookAndFeelException, UserException

Exceptions in Java

Direct Known Subclasses of RunTimeException

ArithmeticException, ArrayStoreException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, EmptyStackException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IndexOutOfBoundsException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NullPointerException, ProfileDataException, ProviderException, RasterFormatException, SecurityException, SystemException, UnsupportedOperationException

`RuntimeExceptions are thrown by the JVM.`
`RuntimeExceptions can be thrown without a declaration.`

User Defined Exception

```
public class UnknownOpException extends Exception
{
   public UnknownOpException()
   {
      super("UnknownOpException");
   }
   public UnknownOpException(char op)
   {
      super(op + " is an unknown operator.");
   }
   public UnknownOpException(String message)
   {
      super(message);
   }
}
```



```
/* Attempts to create a circle with a non-positive radius
will fail.*/
public class Circle {
    public Circle(double r) throws RuntimeException {
        if(r <= 0){
            throw new RuntimeException("The radius
       must be positive: "+r);
        }
        radius = r;
    }
    public double circumference(){
        return 2*Math.PI*getRadius();
    }
    public double area(){
        final double r = getRadius();
        return Math.PI*r*r;
    }
    public double getRadius(){
        return radius;
    }
    private final double radius;
}
```

Exceptions in Java

```
class CircleMain1 {
    public static void main(String[] args)
{// Allow up to 5 circles to be created.
        final int max = 5;
        System.out.println("Please input "+max+"
       radius values.");
        for(int i = 0; i < max; i++){
            double radius = SavitchIn.readLineDouble();
            Circle c = new Circle(radius);
            System.out.println("Area = "+c.area());
        }
    }
}
```


Exceptions in Java

```
public static void main(String[] args){
        SimpleInput keyboard = new SimpleInput();
        // Allow up to 5 circles to be created.
        final int max = 5;
        System.out.println("Please input "+max+" radius
values.");
        for(int i = 0; i < max; i++){
            try{
                double radius = SavitchIn.readLineDouble();
                Circle c = new Circle(radius);
                System.out.println("Area = "+c.area());
            }
            catch(RuntimeException e){
                System.out.println(e.getMessage());
            }
        }
    }
}
```

Exceptions in Java

```
public class CircleMain3 {
    public static void main(String[] args)
    {   // Require exactly 5 circles to be created.
        Circle[] circles = null;
        do{
            try
            {
                final int max = 5;
                circles = createCircles(max);
            }
            catch(RuntimeException e){
                System.out.println(e.getMessage());
          }
        }
        while(circles == null);
        // Got them all.
        for(int i = 0; i< circles.length; i++){
            System.out.print(circles[i].getRadius()+"
             ");
        }
        System.out.println();
    }
```


Exceptions in Java

```
// Return an array containing max Circle objects.
// The array must be returned full, or not at all.
    public static Circle[] createCircles(int max) throws
RuntimeException
   {
        Circle[] circles = new Circle[max];
        System.out.println("Please input "+max+" radius
values.");
        for(int i = 0; i < max; i++){
            double radius = SavitchIn.readLineDouble();
            circles[i] = new Circle(radius);
        }
        return circles;
    }
}
```