F10 - Exceptions

ID1004 Objektorienterad programmering

Fredrik Kilander fki@kth.se

Exception!

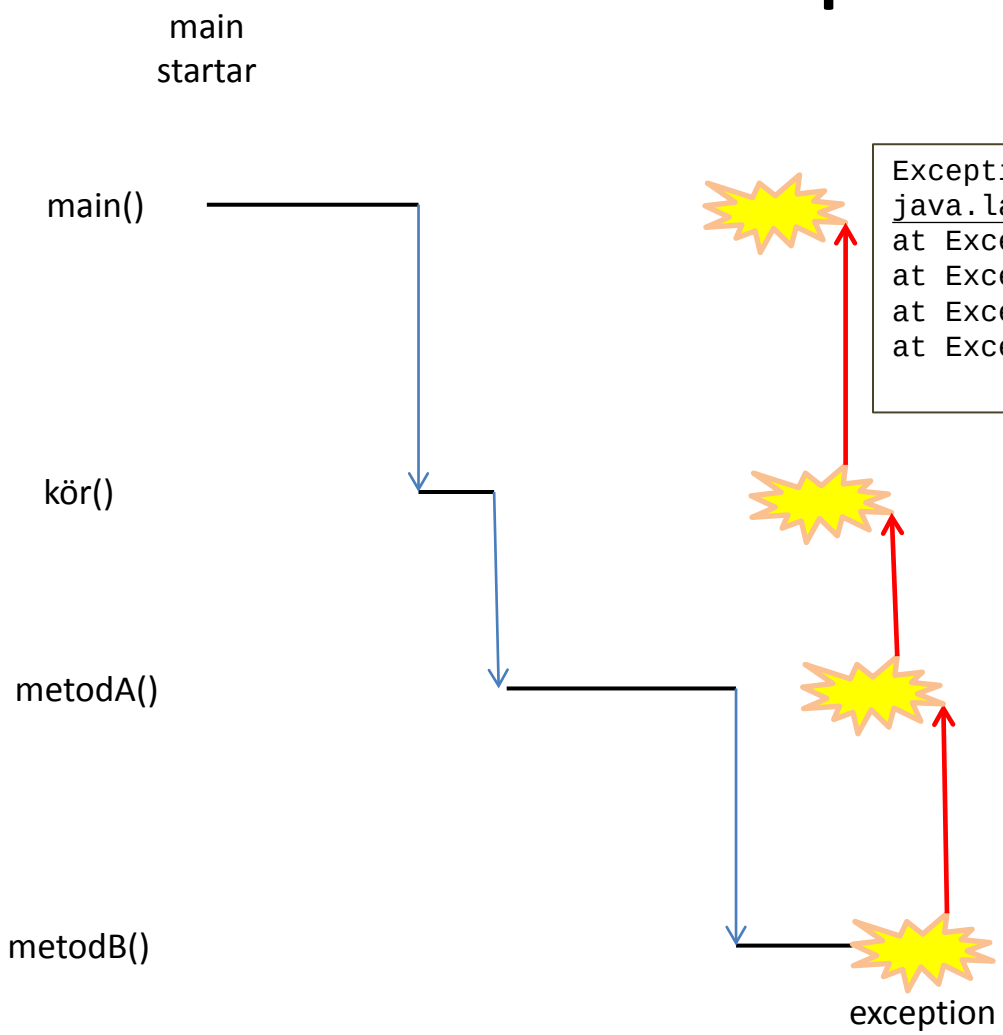
```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at ExceptionDemo.methodB(ExceptionDemo.java:5)
at ExceptionDemo.methodA(ExceptionDemo.java:10)
at ExceptionDemo.kör(ExceptionDemo.java:14)
at ExceptionDemo.main(ExceptionDemo.java:21)
```

Exception!

```
public class ExceptionDemo {

  private void methodB() {
    int n = 123 / 0;
    System.out.println(n);
  }

  private void methodA() {
    methodB();
  }

  public void kör() {
    methodA();
  }
  /**
   * @param args
   */
  public static void main(String[] args) {
    new ExceptionDemo().kör();
  }

}
```

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at ExceptionDemo.methodB(ExceptionDemo.java:5)
at ExceptionDemo.methodA(ExceptionDemo.java:10)
at ExceptionDemo.kör(ExceptionDemo.java:14)
at ExceptionDemo.main(ExceptionDemo.java:21)
```

Exceptions




```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at ExceptionDemo.methodB(ExceptionDemo.java:5)
at ExceptionDemo.methodA(ExceptionDemo.java:10)
at ExceptionDemo.kör(ExceptionDemo.java:14)
at ExceptionDemo.main(ExceptionDemo.java:21)
```

Exceptions (undantag)

- En exception är ett objekt
- En exception kastas uppåt längs anropsstacken när ett fel inträffar
- Om en exception når ut ur metoden main utan att fångas avslutas programmet

Exempel på orsaker

- Division med noll
- Arrayindexering utanför arrayen
- En fil kunde inte hittas
- En I/O-operation kunde inte slutföras normalt
- Programmet försökte referera null
- Programmet försökte bryta mot säkerhetspolicyn

Exempel

```
public class Example10_1 {
  public static void main(String [] args) {
    String s = null;
    int length = s.length();
  }
}
```

```
>java Example10_1
Exception in thread "main" java.lang.NullPointerException
      at Example10_1.main(Example10_1.java:4)

>
```

Exempel

```
public class Example10_2 {
  public static void main(String [] args) {
    int d = 99;
    int n = 0;
    System.out.println(d/n);
  }
}
```

```
>java Example10_2
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at Example10_2.main(Example10_2.java:5)

>
```

Exempel

```
public class Example10_3 {
  private int badCalc(int d, int n){
    return d/n;
  }
  public static void main(String [] args) {
    System.out.println(new Example10_3().badCalc(99, 0));
  }
}
```

```
>java Example10_3
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at Example10_3.badCalc(Example10_3.java:3)
      at Example10_3.main(Example10_3.java:6)

>
```

Vanligt anrop

```
public int[] loadAgeArray(InputStream in){

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

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


  int [] ages = loadAgeArray(System.in);

  double m = average(ages);

  System.out.println(”Avg age: ” + m);
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

nums = [10, 20, 30]
nums.length == 3

Vanligt anrop

```
public int[] loadAgeArray(InputStream in){

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

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


  int [] ages = loadAgeArray(System.in);

  double m = average(ages);

  System.out.println(”Avg age: ” + m);
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

nums = [10, 20, 30]
nums.length == 3

Vanligt anrop

ArithmeticException

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


  int [] ages = loadAgeArray(System.in);

  double m = average(ages);

  System.out.println("Avg age: " + m);
}
```

```
public int[] loadAgeArray(InputStream in){

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

/0

nums = []
nums.length == 0

Hantera exceptions

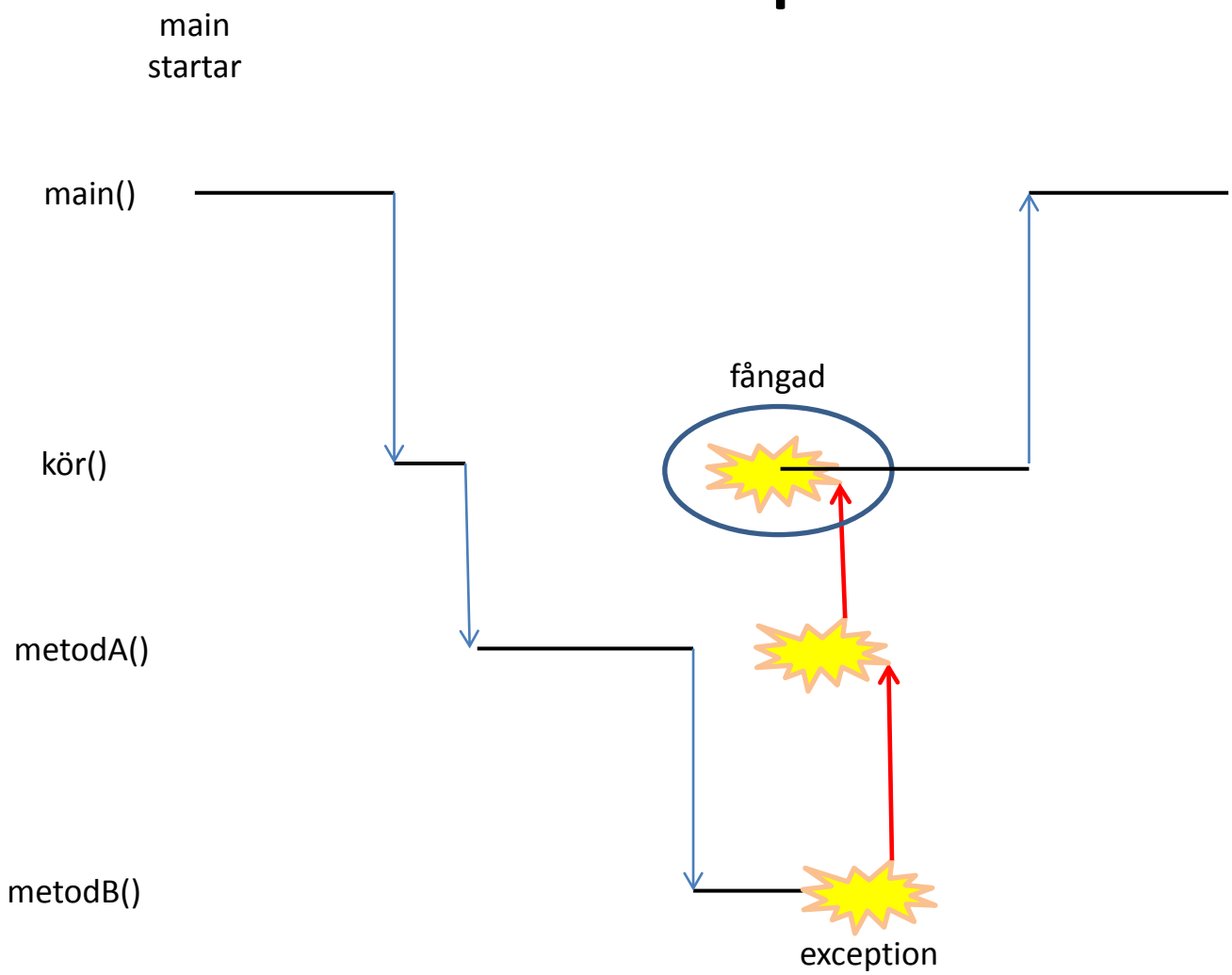
- Vissa typer av exceptions kan förutses

- Designern kan välja att för var och en:
  - inte hantera denna exception
  - hantera den där den kan kastas
  - hantera den högre upp i anropskedjan

Exception!

```
public class ExceptionDemo {

  private void methodB() {
    int n = 123 / 0;
    System.out.println(n);
  }

  private void methodA() {
    methodB();
  }

  public void kör() {
    try {
      methodA();
    } catch (ArithmeticException aex) {

    }
  }
  /**
   * @param args
   */
  public static void main(String[] args) {
    new ExceptionDemo().kör();
  }
}
```

Exceptions



Fånga exceptions med try-catch

Exakt 1 try-klausul

Oftast minst en catch

finally ute-lämnas ofta

```
try {

  /* kod som kan kasta exceptions */

}
catch ( exceptionklass variabel ){
  /* hantera en eller flera exceptions */
}
finally {
  /* kod här utförs alltid, oavsett om en
     exception kastades eller ej */
}
```

Fånga exceptions med try-catch

Exakt 1 try-klausul

Oftast minst en catch

finally ute-lämnas ofta

```
try {

  int n = s.length();

}
catch (NullpointerException npx){
  npx.printStackTrace();
}
finally {
  /* kod här utförs alltid, oavsett om en
     exception kastades eller ej */
}
```

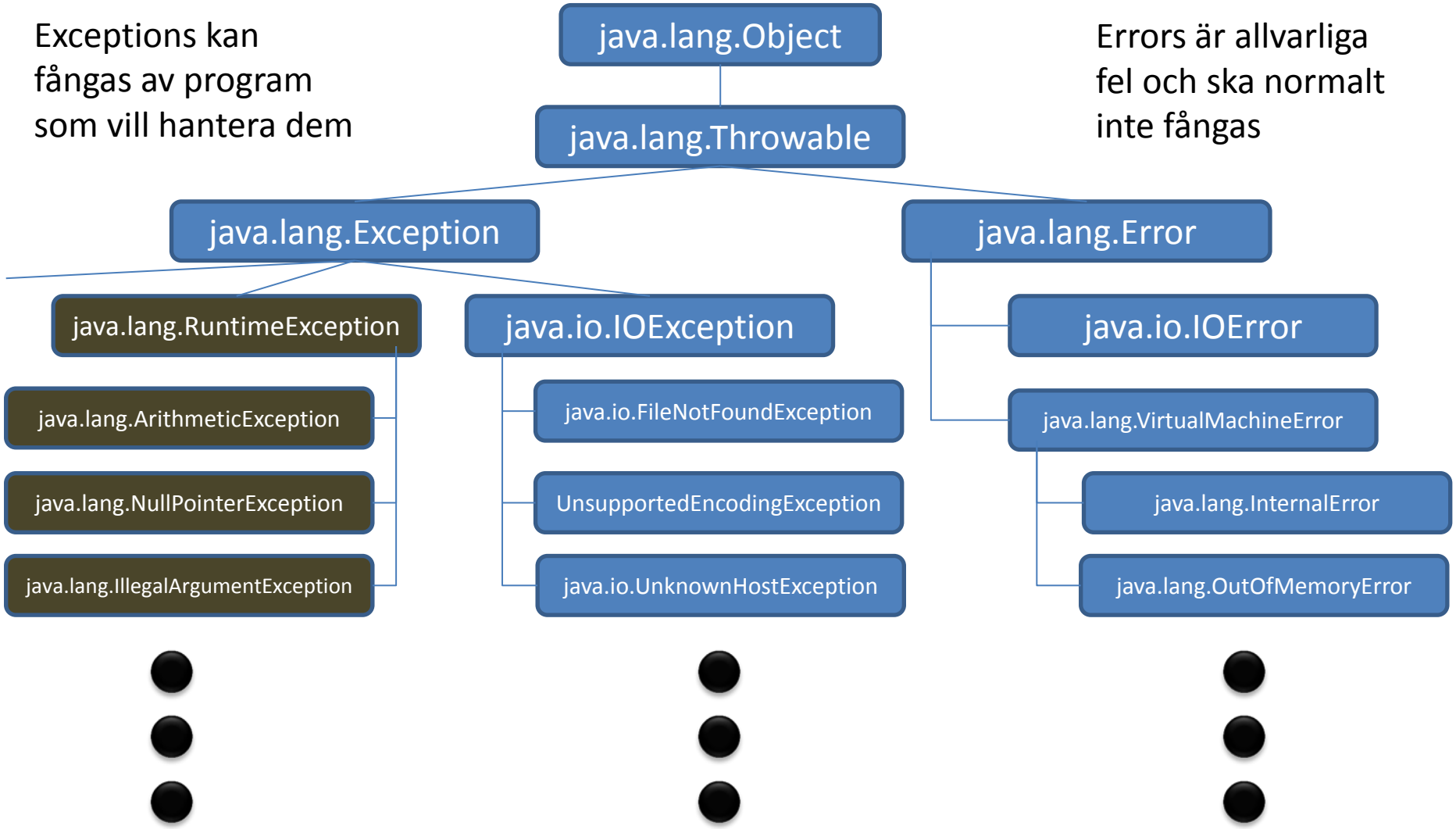
Fånga exceptions med try-catch

```
Datasource database = ...;
Connection myConnection = null;
try {
  myConnection =
    dabatase.getConnection(myName, myPassword);
  // Prata med databasen
  ...
}
catch (SQLException sqx){
  System.err.println(sqx.getMessage());
}
finally {
  if (myConnection != null) {
    myConnection.close();
  }
}
```

Exceptions är ordnade i klasser

Exceptions kan fångas av program som vill hantera dem

Errors är allvarliga fel och ska normalt inte fångas

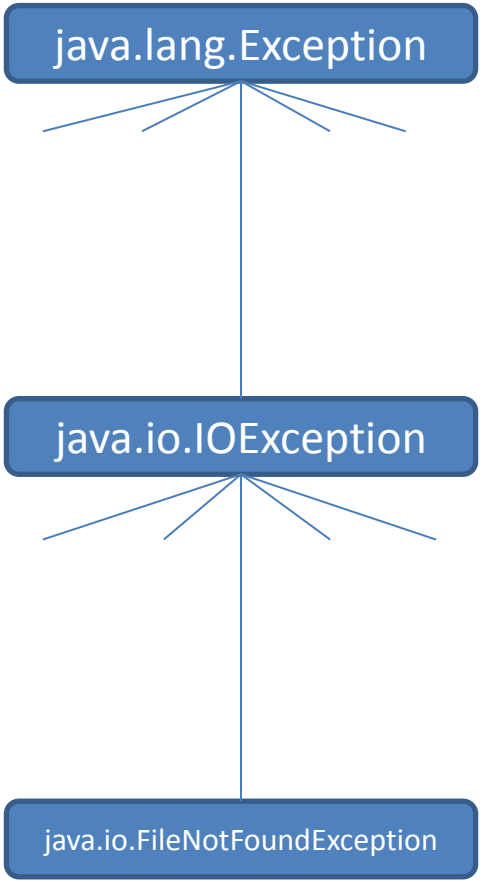


catch med klass på rätt nivå

```
try {

}
catch (java.lang.Exception ex) {
  // fångar alla exceptions
}
```

```
try {

}
catch (java.io.IOException iox) {
  // fångar alla IO-exceptions
}
```

```
try {

}
catch (java.io.FileNotFoundException fnx) {
  // fångar FileNotFoundException
}
```



Händiga metoder i en exception

- String getMessage()
  - kort beskrivning
- printStackTrace()
  - skriver ut hela spåret på standard error

```
try {
  FileReader fr = new FileReader(”Names.txt”);
  … // Read names from file
  fr.close();
} catch (java.io.FileNotFoundException fnx) {
  System.out.println (”Oops, ” + fnx.getMessage());
}
```

try-catch example: getMessage()

```
public class Example10_4 {
  private int badCalc(int d, int n){
    return d/n;
  }
  public static void main(String [] args) {
    Example10_4 ex = new Example10_4();
    try {
      ex.badCalc(99, 0);
    }
    catch (ArithmeticException arx) {
      System.out.println("Message is " + arx.getMessage());
    }
  }
}
```

```
>java Example10_4
Message is / by zero

>
```

try-catch example: printStackTrace()

```
public class Example10_5 {
  private int badCalc(int d, int n){
    return d/n;
  }
  public static void main(String [] args) {
    Example10_5 ex = new Example10_5();
    try {
      ex.badCalc(99, 0);
    }
    catch (ArithmeticException arx) {
      arx.printStackTrace();
    }
  }
}
```

```
>java Example10_5
java.lang.ArithmeticException: / by zero
      at Example10_5.badCalc(Example10_5.java:3)
      at Example10_5.main(Example10_5.java:8)

>
```

Måste man fånga exceptions?

- Nej, inte om det är ok att programmet avbryts
- Men: en metod kan deklarera att den kastar en eller flera exceptions
- Om man anropar en sådan metod måste man antingen
  - fånga dem, eller
  - deklarara att de kan komma att kastas
- Detta gäller inte RuntimeException och dess underklasser

java.util.Scanner.nextInt()

nextInt

public int **nextInt**()

Scans the next token of the input as an int. An invocation of this method of the form nextInt() behaves in exactly the same way as the invocation nextInt(radix), where radix is the default radix of this scanner.

Returns: the int scanned from the input

Throws:
InputMismatchException - if the next token does not match the *Integer* regular expression, or is out of range
NoSuchElementException - if input is exhausted
IllegalStateException - if this scanner is closed

Dessa tre är subklasser till java.lang.RuntimeException. De behöver inte fångas eller deklareras som kastbara. Men det är tillåtet att göra bägge.

Läsa tal med felhantering

```
Scanner scan = new Scanner(System.in);

int num;
boolean done = false;

while (!done) {
  try {
    System.out.print(”Ange längd:”);
    num = scan.nextInt();
    done = true;
  }
  catch (InputMismatchException imx) {
    scan.nextLine();
    System.out.println(”Skriv ett tal”);
  }
}
```

Exception propagation

- Det är inte alltid självklart på vilken nivå en exception hanteras bäst
- Förutsägbara fel bör hanteras där de kan åtgärdas
- Mindre förutsägbara fel kanske inte ska fångas alls (det kan dölja buggar)

Vanligt anrop

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

  File inFile = new File(args[0]);

  int [] ages = loadAgeArray(inFile);

  double m = average(ages);

  System.out.println(”Avg age: ” + m);
}
```

```
public int[] loadAgeArray(File in)
  throws FileNotFoundException
{

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

Vanligt anrop

```
public int[] loadAgeArray(File in)
  throws FileNotFoundException
{

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

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

  File inFile = new File(args[0]);

  int [] ages = loadAgeArray(inFile);

  double m = average(ages);

  System.out.println(”Avg age: ” + m);
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

loadAgeArray kastar FileNotFoundException
Detta får inte ignoreras i main.

Vanligt anrop

```
public int[] loadAgeArray(File in)
  throws FileNotFoundException
{

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

```
public static void main(String[] args)
  throws FileNotFoundException
{

  File inFile = new File(args[0]);

  int [] ages = loadAgeArray(inFile);

  double m = average(ages);

  System.out.println(”Avg age: ” + m);
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

loadAgeArray kastar FileNotFoundException
Detta får inte ignoreras i main
Deklarera i main också

Vanligt anrop

ArrayIndexOutOfBoundsException

```
public static void main(String[] args)
  throws FileNotFoundException
{

  File inFile = new File(args[0]);

  int [] ages = loadAgeArray(inFile);

  double m = average(ages);

  System.out.println(”Avg age: ” + m);
}
```

```
public int[] loadAgeArray(File in)
  throws FileNotFoundException
{

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

Tänkbara scenarior

Vanligt anrop

FileNotFoundException

```
public static void main(String[] args)
  throws FileNotFoundException
{

  File inFile = new File(args[0]);

  int [] ages = loadAgeArray(inFile);

  double m = average(ages);

  System.out.println("Avg age: " + m);
}
```

```
public int[] loadAgeArray(File in)
  throws FileNotFoundException
{

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

Tänkbara scenarior

Vanligt anrop

ArithmeticException

```
public static void main(String[] args)
  throws FileNotFoundException
{

  File inFile = new File(args[0]);

  int [] ages = loadAgeArray(inFile);

  double m = average(ages);

  System.out.println("Avg age: " + m);
}
```

```
public int[] loadAgeArray(File in)
  throws FileNotFoundException
{

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

Tänkbara scenarior

Vanligt anrop

```
public static void main(String[] args) {
  File inFile = null;
  if (0 < args.length) {
    inFile = new File(args[0]);
  else {
    System.out.println(”Filename expected”);
    return;
  }

  int [] ages;
  try {
    ages = loadAgeArray(inFile);
  }
  catch (FileNotFoundException fex) {
    System.out.println(”File not found”);
    return;
  }

  if (0 < ages.length) {
    double m = average(ages);
    System.out.println(”Avg age: ” + m);
  }
  else {
    System.out.println(”No ages read from file”);
  }
}
```

```
public int[] loadAgeArray(File in)
  throws FileNotFoundException
{

  Scanner scan = new Scanner(in);

  // Läs in och lägg i en ny array

  ...

  return ar;
}
```

```
public double average(int [] nums){

  double sum = 0d;

  for (int n : nums) {
    sum += n;
  }

  double avg = sum / nums.length;

  return avg;
}
```

Egna exceptions

```
public void setDegrees(float degrees)
  throws IllegalArgumentException
{
  if ((degrees < 0) || (360 <= degrees)) {
    throw new IllegalArgumentException(”Degree out of range”);
  }
  ...
}
```

Man kan använda en exceptionklass som redan finns
eller skapa en egen.

Egna exceptions

```
public class EngineException extends Exception {}
```

```
public class ElectricalException extends EngineException {}
```

```
public class CircuitException extends ElectricalException {}
```

```
public class PowerException extends ElectricalException {}
```

```
public class SensorException extends EngineException {}
```

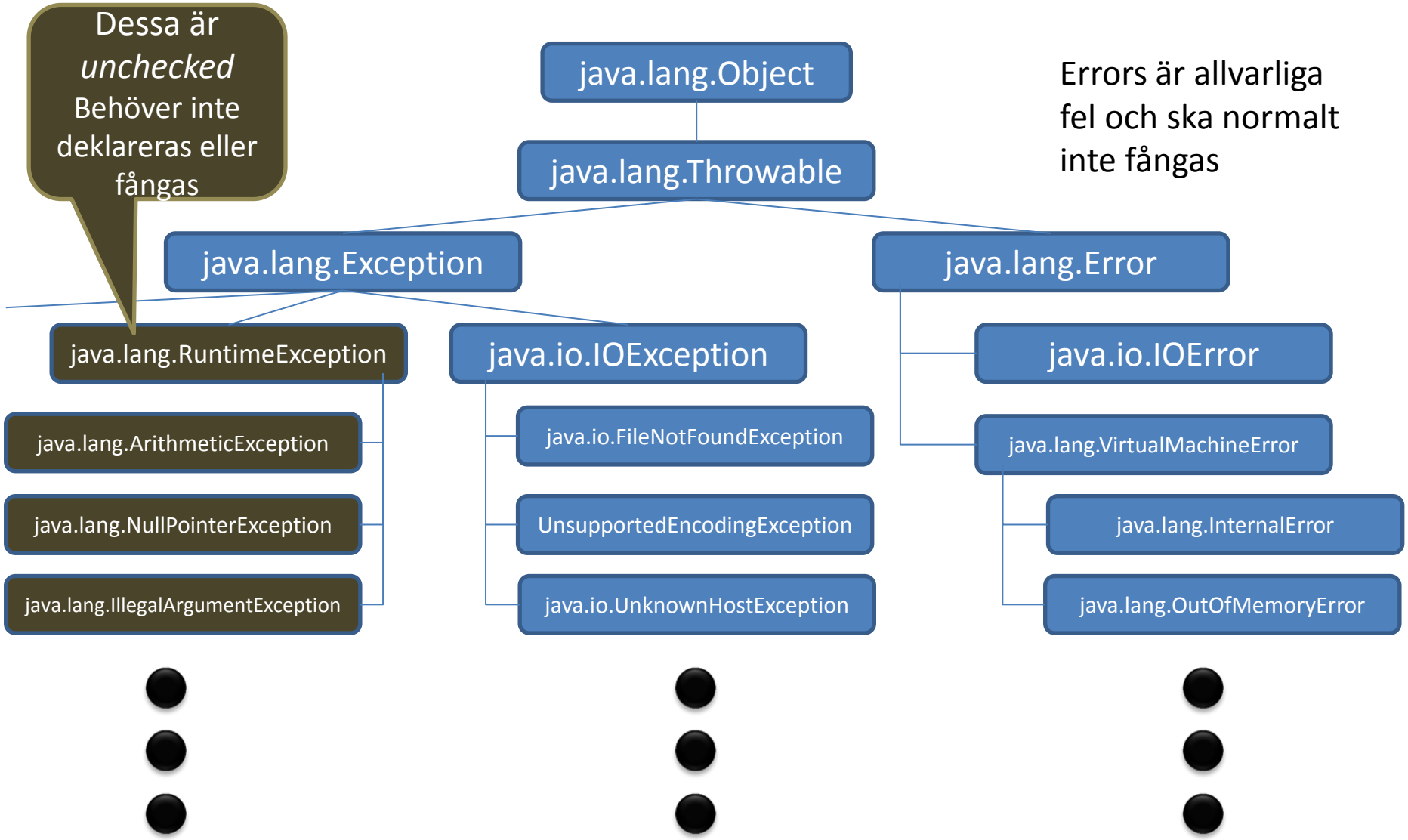
```
public class SensorOffRangeException extends SensorException {}
```

```
public class SensorOffRangeHighException
  extends SensorOffRangeException {}
```

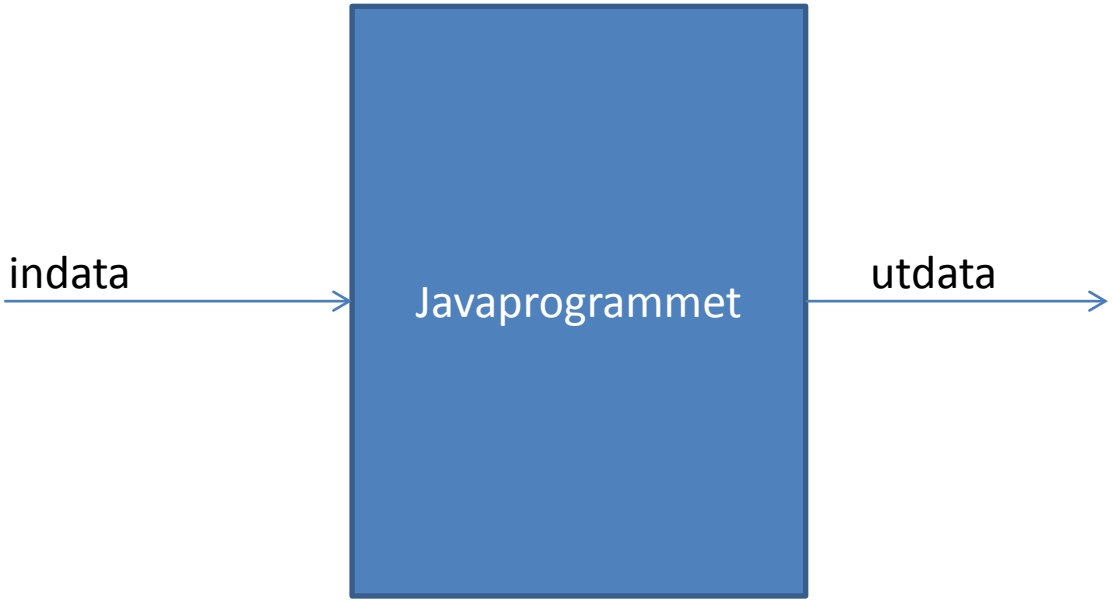
```
public class SensorOffRangeLowException
  extends SensorOffRangeException {}
```

```
public class SensorUnreadableException extends SensorException {}
```

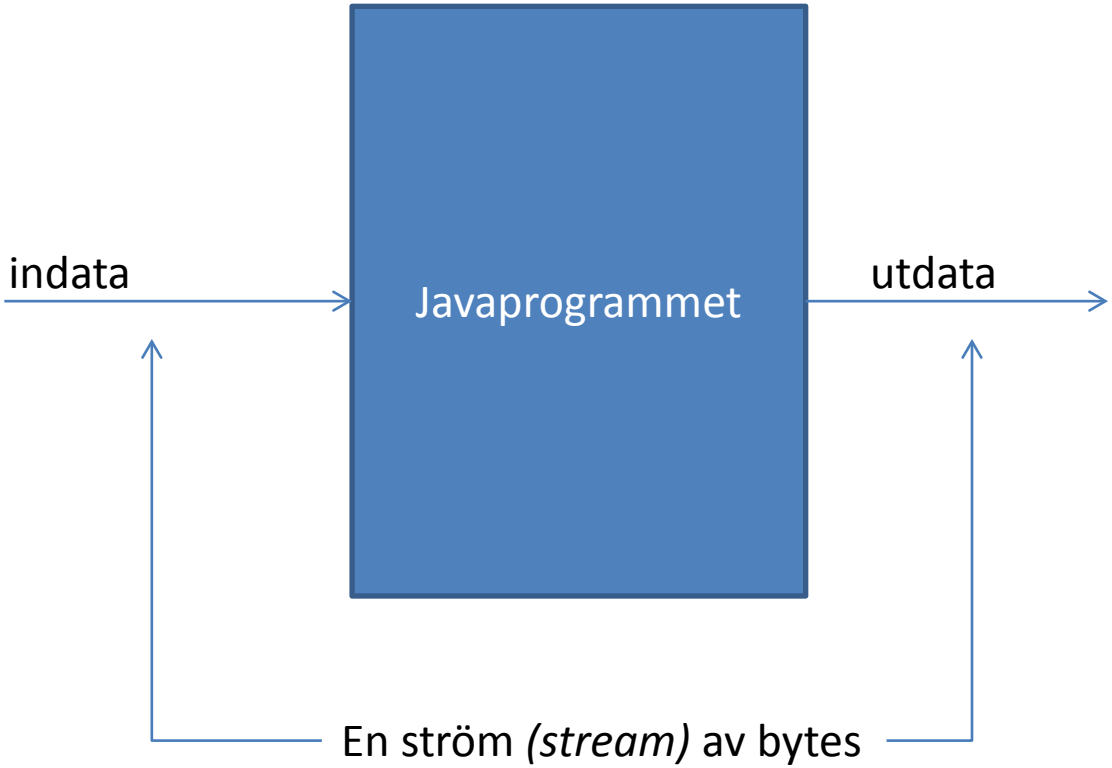
Checked - unchecked



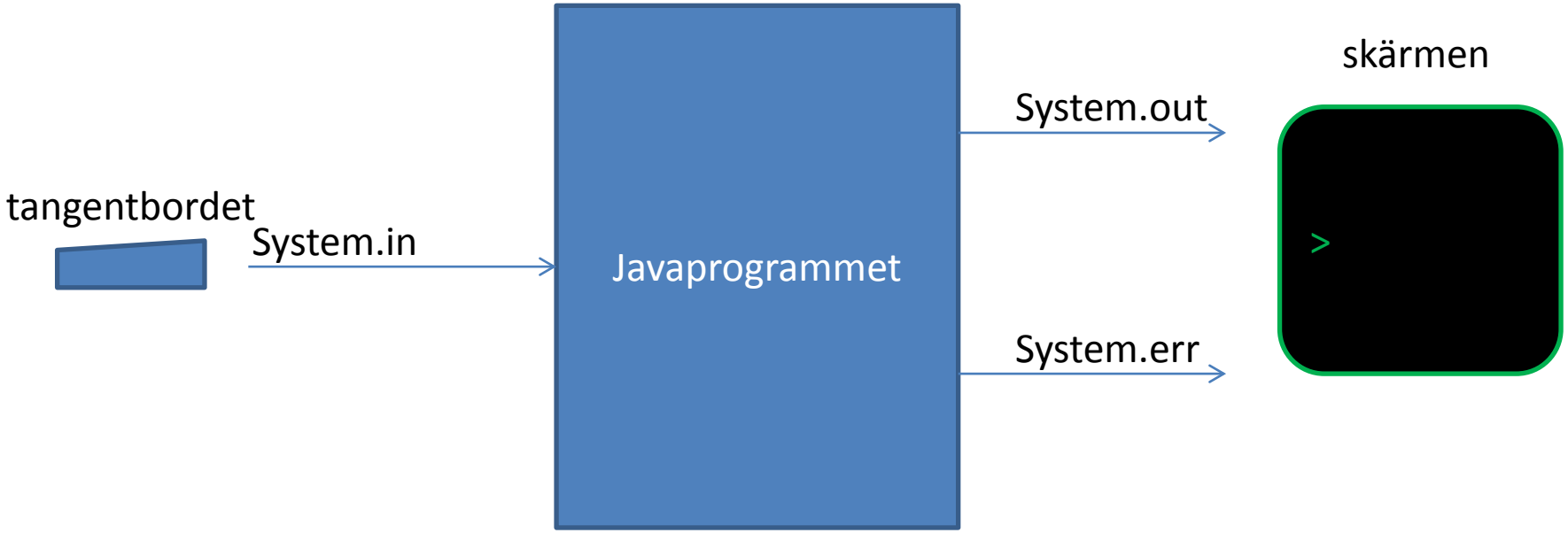
in- och utmatning (I/O)



in- och utmatning (I/O)

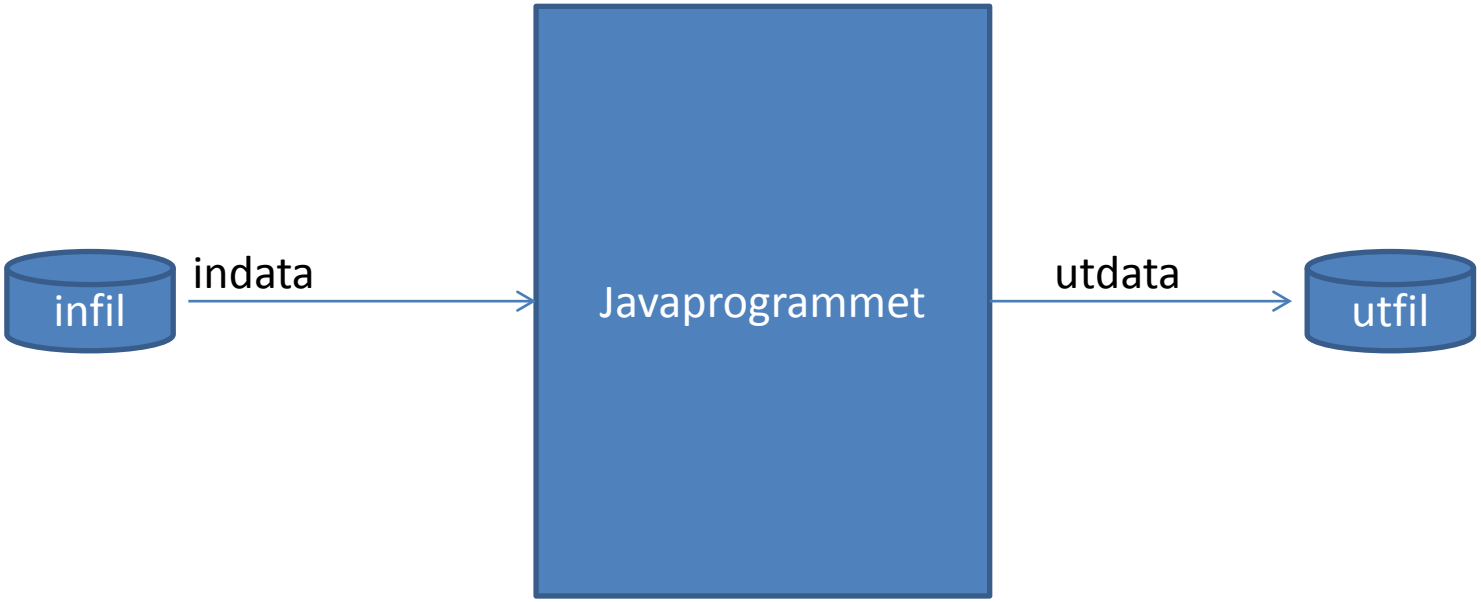


in- och utmatning (I/O)



Dessa streams finns med från början

in- och utmatning (I/O)

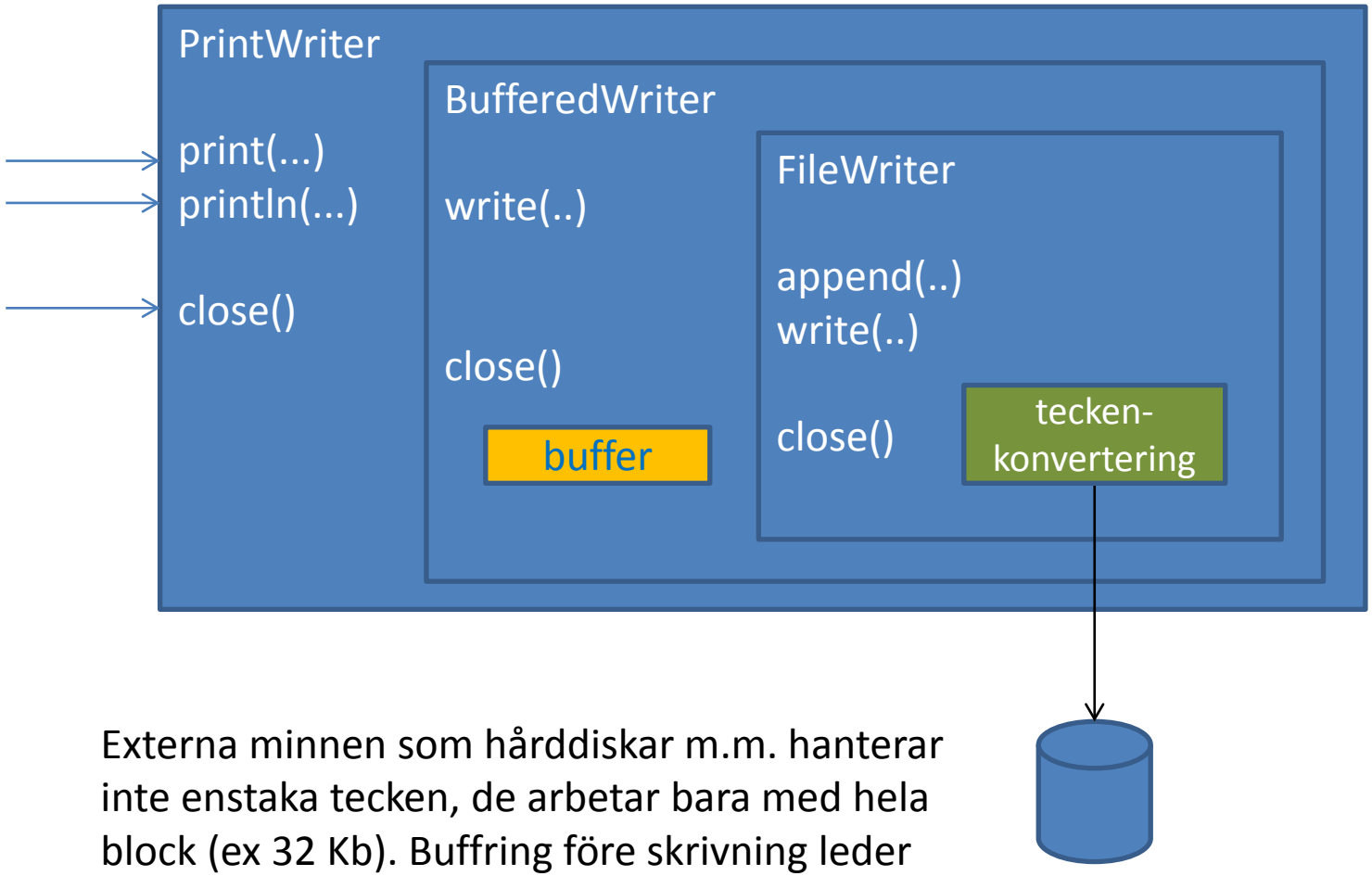


Läsning och skrivning med filer görs med hjälp av andra streams
Programmeraren måste själv beställa dessa

Exempel på I/O till fil

```
import java.util.Random;
import java.io.*;

public class TestData {
  public static void main(String [] args) throws IOException {
    Random rnd = new Random();
    String outFileName = ”test.dat”;
    // En FileWriter konverterar text till det lokala systemet
    FileWriter fw = new FileWriter(outFileName);
    // En BufferedWriter gör skrivningen till filsystemet effektivare
    BufferedWriter bw = new BufferedWriter(fw);
    // En PrintWriter har bra metoder för att skriva ut med
    PrintWriter pw = new PrintWriter(bw);
    // Skriv ut 500 rader med 30 slumptal i varje rad
    for (int line = 0; line < 500; line++) {
      for (int item = 0; item < 30; item++) {
        pw.print(rnd.nextInt(100) + ” ”);
      }
      pw.println();
    }
    pw.close(); // Se till att buffrat data skrivs till disk.
  }
}
```

Text, buffring och metoder



Externa minnen som hårddiskar m.m. hanterar inte enstaka tecken, de arbetar bara med hela block (ex 32 Kb). Buffring före skrivning leder till färre skrivningar i filen.

inga undantag

SLUT FÖR DENNA GÅNG