Exception

- Är en exceptionell händelse som inträffar under exekvering av ett program och som avbryter det normala programflödet
- Kan uppstå av flera orsaker som t.ex.:

  Hårdvaruproblem

  Överbelastning

  Vanliga programbuggar
- När en sådan händelse inträffar sägs det att ett Exception "kastas" (eng. throws)
- Den metod som förväntas hantera ett Exception sägs "fånga" (eng. catch) ett Exception
- Kapslar in orsaken till den exceptionella händelsen i ett objekt

Kodstruktur för Exception-hantering - try / catch

- För att kunna hantera kod som kan kasta Exceptions skrivs koden inom ett s.k. try-block
- För att fånga ett visst slag av Excpetion används ett till flera catch-block

  Ex:

```
try{
//"Riskabel" kod som kan kasta en eller flera olika sorters Exceptions

} catch (ExceptionTypeOne e1) {
//Fångar eventuella Exception av typ ExceptionTypeOne

} catch (ExceptionTypeTwo e2) {
//Fångar eventuella Exception av typ ExceptionTypeTwo
}
```

Exception-hantering pseudokod

- Ett Exception skulle kunna kastas om t.ex. en nätverksresurs inte är tillgänglig:

```
try {
//Ladda en egenskapsfil från nätverket

} catch (CantReadFromNetworkException e) {

//Använd en lokal fil istället
}
```

Här ges det möjlighet att använda en lokal resurs istället. Utan Exception-hantering skulle denna kod inte fungera

Kodstruktur för Exception-hantering - finally

- Används för att ge möjlighet att "städa upp" eventuella resurser även om ett Exception kastas
- finally-blocket garanteras att alltid exekveras (förutom under vissa extrema situationer)
- finally exekveras oavsett om ett Exception kastas eller inte

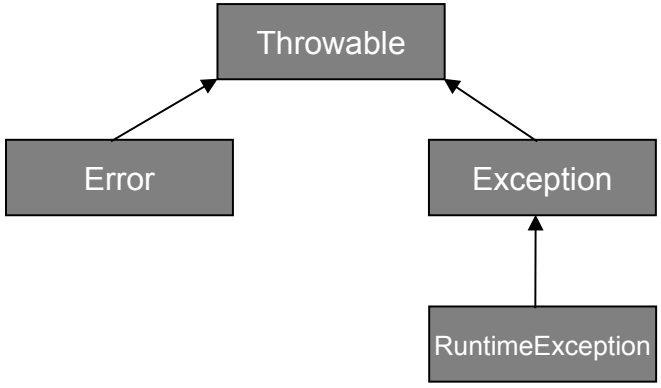
```
try{
//Kod som öppnar en koppling till databasen
} catch (Exception e) {
//Fångar eventuella Exceptions

} finally {
//Stäng ner eventuella öppna kopplingar till databasen
}
```

Flöde:

1. Koden i try-blocket exekveras
2. Om ett Excepetion kastas exekveras koden i catch-blocket
3. Oavsett om ett Exception kastas eller inte exekveras koden i finally-blocket

Exception - hierarki

- Alla Exception är subklasser till klassen **Throwable**
- Från Throwable utgår två subklasser: **Error** och **Exception**
- Från dessa utgår sedan flera subklasser
- Klasser som har Error som superklass (indirekt eller direkt) representerar exceptionella händelser som vanligtvis är så allvarliga att de inte går att återhämta sig från
- Exception representerar en exceptionell händelse som inte är resultatet av ett programfel utan att t.ex. en nätverksresurs inte är tillgänglig
- **RuntimeException** är en subklass till Exception och representerar programfel som t.ex. att försöka hämta ett element i en Array utanför dess indexgräns



Hämta information från ett Exception

- Varje Exception har metoden **printStackTrace()** som skriver ut var detta Exception uppstod
- Metoden **getMessage()** returnerar ett meddelande om detta Exception

  Ex:

```
try{
//Kod som kan kasta ett Exception

} catch (Exception e) {
   e.printStackTrace;
   System.out.println(e.getMessage());
...
}
```

Checked och Unchecked -Exceptions

- Exception som är subklasser till Error eller RuntimeException behöver inte fångas. Dessa kallas **"Unchecked Exceptions"**

- Subklasser till Exception måste däremot fångas eller kastas vidare. Dessa kallas **"Checked Exceptions"**

- Anledningen till detta är att det vanligtvis bara går att återhämtas från Checked Exceptions

  Ex:

```
//Metod som kastar ett Unchecked Exception
public void printMessage(String message){
  if(message == null){
                throw new NullPointerException("Argument must not be null.");
  }
}

//Anrop av metod som kastar ett Unchecked Exception
printMessage(null);
```

  Kod som anropar printMessage behöver inte göra det i ett try/catch -block. Det Exception som printMessage kastar är ett Unchecked Exception

Fånga Exceptions - att tänka på

- Ett catch-block fångar bara det Exception som det anger i sitt metodhuvud samt subklasser till den Exception-typen
- Ordningen på catch-blocken är viktiga
- Den mest specifika typen av Exception skall fångas först

Ex:

```
try {
//Kod som kan kasta Exception av olika slag
} catch (Exception e) {
        // Detta fångar alla typer av Exceptions
} catch (IOException e) { ← Detta ger kompileringsfel
        // Finns inga Exception kvar att fånga, denna kod körs aldrig
}
```

Det andra catch-blocket kommer aldrig att kunna nås eftersom alla Exceptions redan är fångade. Detta eftersom IOException är subklass till Exception

Metoder som kan kasta Exception

- Om en metod kan kasta ett Checked Exception måste detta deklareras. Detta görs genom att ange detta i metodhuvudet

- En metod kan kasta fler än en typ av Exceptions

```
//Metod som deklarerar att den kastar IOException
public File getFile(String filename) throws IOException {
        File file = new File(filename);
        file.createNewFile();
        return file;
}

//Metod som anropar getFile() måste ta hand om det Exception som kan kastas
public void readFromFile(){
  try {
        File file = getFile("storage.txt");
   } catch (IOException e) {
        e.printStackTrace();
   }
}

--- Alternativt kasta det vidare

public void readFromFile() throws IOException {
  File file = getFile("storage.txt");
}
```

Skapa egna Exceptions

- För att skapa ett eget Exception skapar man en subklass till Exception (eller någon av Exceptions subklasser)

  Ex:

```
//Skapar en subklass till Exception
public class MyException extends Exception{

  public MyException(String message){
            super(message);
        }
}

//Metod som kan kasta MyExcepetion
public String reverse(String string) throws MyException {

  //Denna metod kan kasta MyException
  throw new MyException("Exception....");
}
```

Allmänt om Excpetions

- **(Item 57)**
- Använd bara för exceptionella händelser, **aldrig** för kontrollering av programflöde:

  Ex:

```
try {
  int i = 0;
  while(true){
        System.out.println(a[i++]);
  }
} catch (ArrayIndexOutOfBoundsException e) {
//Struntar i Excpetion
}
```

Allmänt om Excpetions forts.

- **(Item 58)**
- Använd Runtime Excpetions för programmeringsfel

  Ex:

```
public void addToStorage(Storable objectToStore) {

if (objectToStore == null) {
  throw new NullPointerExcpetion("'objectToStore' must not be null.");
}
}
```

Allmänt om Excpetions forts.

- **(Item 58)**
- Använd **checked** Exceptions för tillstånd som går att återhämta sig ifrån

  Ex:
  try {
  //Ladda en egenskapsfil från nätverket

  } catch (CantReadFromNetworkException) {

  //Använd en lokal fil istället
  }

Allmänt om Excpetions forts.

- **(Item 65)**
- Bortse inte från Excpetion

  Ex:

```
try {
//Kod som kan kasta ett Exception
} catch (Excpetion e){
//Struntar i Exception som fångats
}
```

Övning

1. Skapa ett nytt projekt i Eclipse

2. Skapa klassen **User** som har instansvariablerna: **userId**, **username**, **password**

3. Skapa klassen **UserStorage** som lagrar user-objekt i en array. Denna klass har en metod **addUserToStorage()** som lagrar user-objekt i sin array samt metoden **getUserFromStorage()** som returnerar eventuellt lagrat User-objekt med det **userId** som skickas som argument

4. Subklassa klassen Excepetion för att skapa ditt egna Exception som ska heter **StorageException**

5. Låt metoden **addUserToStorage()** kasta **StorageException** om ett User-objekt med samma userId redan finns lagrad i arrayn

6. Låt metoden **getUserFromStorage()** kasta **StorageException** om det inte finns något User-objekt med angivet userId