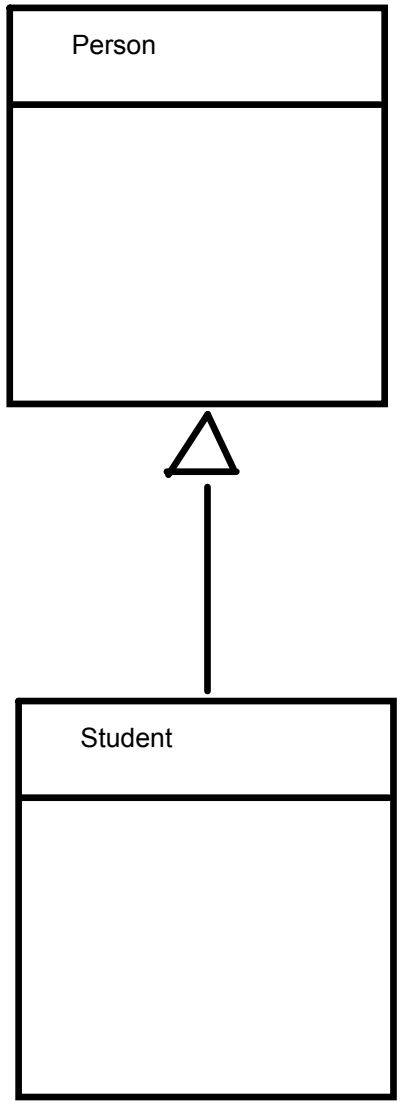
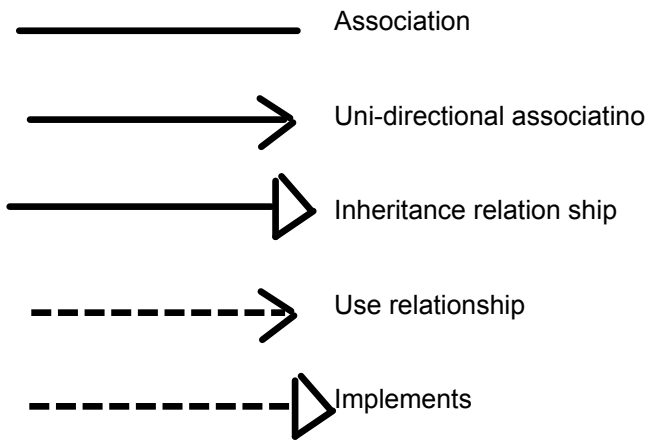




What is inherited

- public and protected method
- public and protected attributes / fields /

What is not inherited

- private methods
- private attributes / fields
- constructors

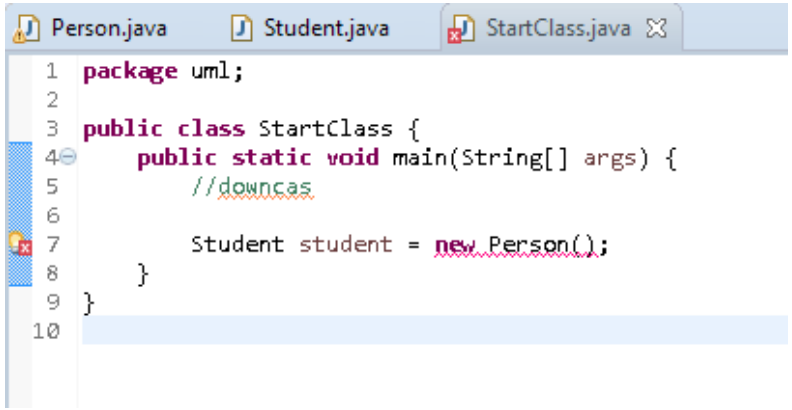
If the super class has a non-trivial constructor, the inheriting class (sub class) also must have a constructor that is calling the constructor of the super class.

If a method of a super class may not be overwritten it has to be declared as final.

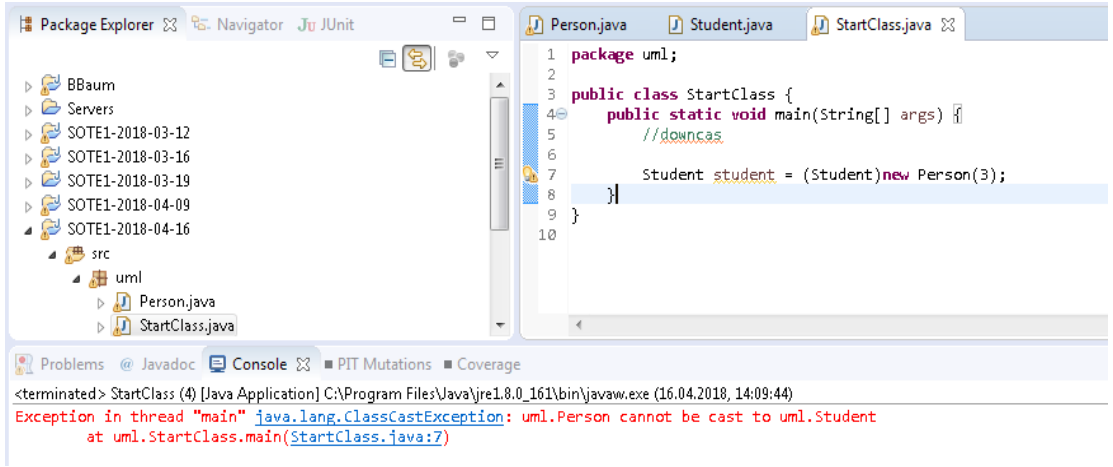
If a method of a super class must be overwritten, it must be declared as abstract. This means that the entire super class must be abstract.

Casting

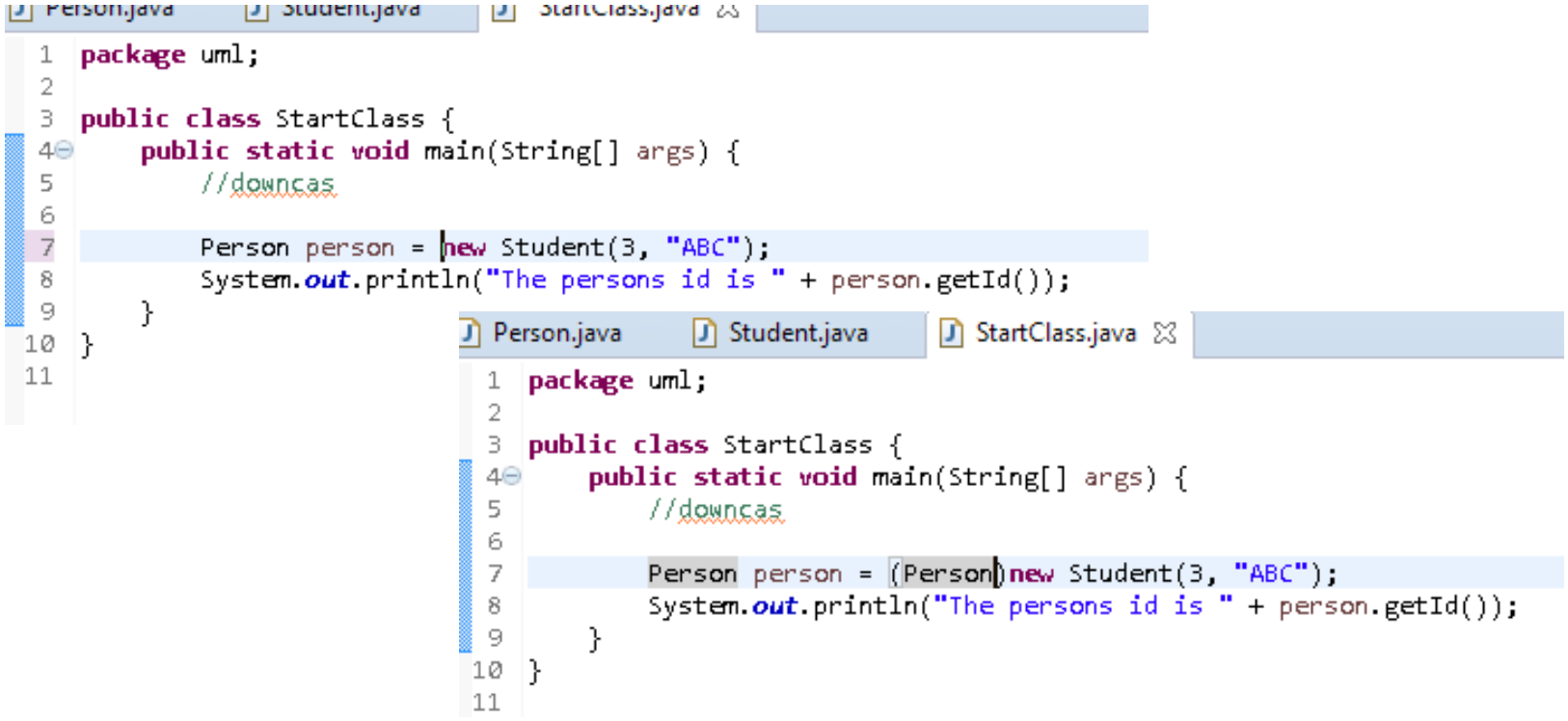
It is not possible to implicitly downcast from one class to another:



It is also not possible to explicitly downcast: We run into a class cast exception



An implicit and an explicit upcast is possible:



Attention: Even if there is an explicit upcast, still the method of the sub class (Student) is invoked!!

Comparision of interfaces and abstract classes

Commonalities

- both have at least one abstract methods
- abstract methods always have to be overwritten
- Both can NOT be instatiated

Differences

- Interfaces only have abstract methods (exception: default methods in Java 8+) whereas abstract methods may have non-abstract methods.
- Interfaces do not use the modifier "abstract"
- It is possible to impement multiple interfaces but inherit from just one class
- Interfaces may not have fields / attributes (until Java 7)

Error handling

Java enforces the error handling

StartError.java

```
1  package errors;
2
3  import java.io.File;
4  import java.io.FileWriter;
5
6  public class StartError {
7      public static void main(String[] args) {
8
9          FileWriter writer = new FileWriter(new File("C:\\tmp.txt"));
10
11     }
12 }
13
```

1. Option to deal with an error: Catch error and deal with it

*StartError.java

```
1  package errors;
2
3  import java.io.File;
4  import java.io.FileWriter;
5  import java.io.IOException;
6
7  public class StartError {
8      public static void main(String[] args) {
9
10         try {
11             FileWriter writer = new FileWriter(new File("C:\\tmp.txt"));
12         } catch (IOException e) {
13             // TODO Auto-generated catch block
14             e.printStackTrace();
15         }
16
17     }
18 }
19
```

2. Option: throw error to calling method or out of main

*StartError.java

```
1  package errors;
2
3  import java.io.File;
4  import java.io.FileWriter;
5  import java.io.IOException;
6
7  public class StartError {
8      public static void main(String[] args) throws IOException {
9
10         FileWriter writer = new FileWriter(new File("C:\\tmp.txt"));
11
12     }
13 }
14
```

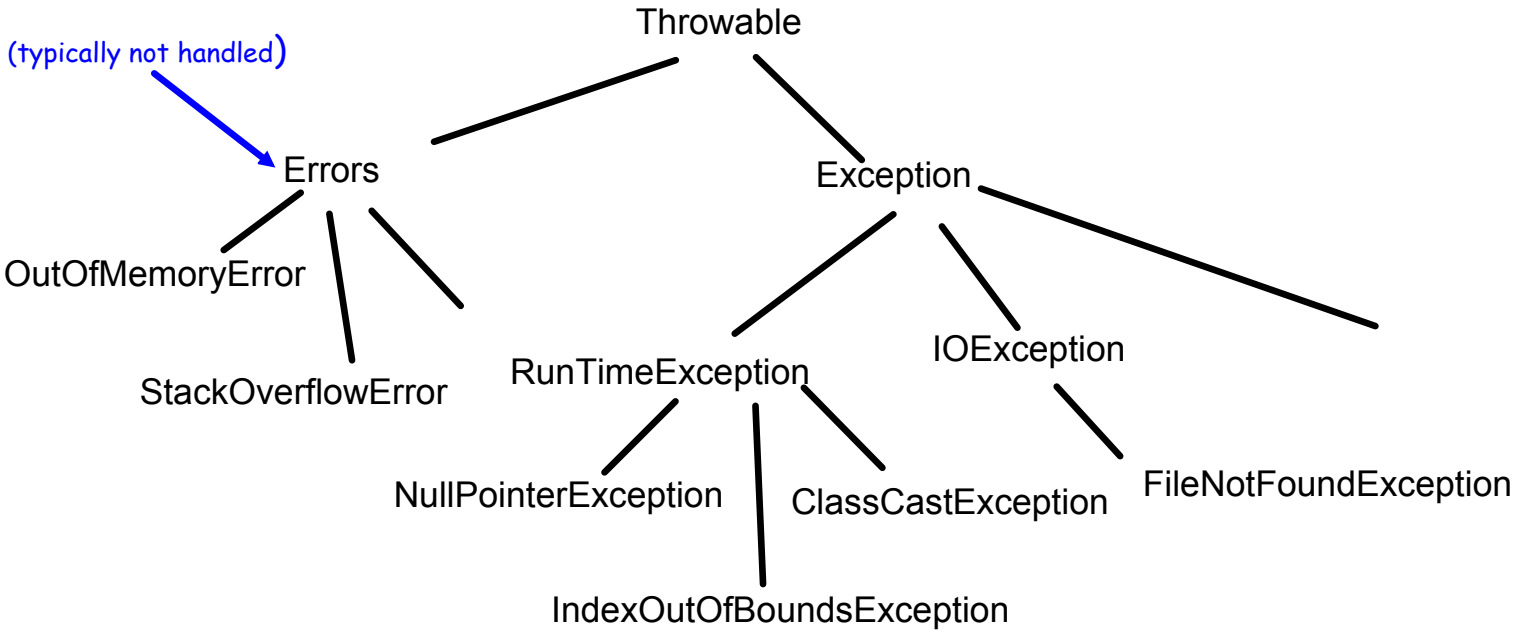
Hierarchy of classes

```
public class StartError {
    public static void main(String[] args) throws IOException {

        FileWriter writer = new FileWriter(new Fil

    }
}
```

IOException - java.io
Exception - java.lang
Throwable - java.lang

thrown by JVM

(typically not handled)



RunTimeException are special: The compiler does not enforce to deal with these exceptions

Overview Package Class Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

Class RuntimeException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AnnotationTypeMismatchException, ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DataBindingException, DOMException, EmptyStackException, EnumConstantNotPresentException, EventException, FileSystemAlreadyExistsException, FileSystemNotFoundException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, IllformedLocaleException, ImagingOpException, IncompleteAnnotationException, IndexOutOfBoundsException, JMRuntimeException, LSException, MalformedParameterizedTypeException, MirroredTypesException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NoSuchMechanismException, NullPointerException, ProfileDataException, ProviderException, ProviderNotFoundException, RasterFormatException, RejectedExecutionException, SecurityException, SystemException, TypeConstraintException, TypeNotPresentException, UndeclaredThrowableException, UnknownEntityException, UnmodifiableSetException, UnsupportedOperationException, WebServiceException, WrongMethodTypeException

---

```
public class RuntimeException
extends Exception
```

`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

`RuntimeException` and its subclasses are *unchecked exceptions*. Unchecked exceptions do *not* need to be declared in a method or constructor's `throws` clause if they can be thrown by the execution of the method or

Import rules for dealing with exception

- don't catch errors, there is not much you can do about it
- catch exceptions as close to the root cause as possible, don't throws exception unless there is no alternative
- never write an empty catch block (you won't find the error)
- catch the exception as specific as possible e.g. catch FileNotFoundException and not just Exception
- also catch-block can contain bugs that have to be handled

You can write your own exception...

StartError.java SoteException.java

```java
package errors;

public class SoteException extends Exception {
    private String filepath;

    public SoteException(String filepath) {
        super();
        this.filepath = filepath;
    }

    public String getFilepath() {
        return filepath;
    }

    public void setFilepath(String filepath) {
        this.filepath = filepath;
    }
}

```

... and use it

StartError.java SoteException.java StartUsingOurException.java

```java
package errors;

public class StartUsingOurException {
    public static void main(String[] args) {

        StartUsingOurException instance = new StartUsingOurException();

        try {
            instance.divide(2, 3);
        } catch (SoteException e) {
            e.printStackTrace();
        }

    }

    public int divide(int a, int b) throws SoteException {
        System.out.println("We are in myMethod");

        if (b == 0) {
            throw new SoteException("C:\\fake");
        }

        return a / b;
    }
}

```

Combination

- try-catch
- try-catch-finally
- try-catch-catch

```java
package errors;

public class StartUsingOurException {
    public static void main(String[] args) {

        StartUsingOurException instance = new StartUsingOurException();

        try {
            instance.divide(2, 3);
        } catch (SoteException e) {
            e.printStackTrace();
        } finally {
            System.out.println("Program ends ");
        }
        System.out.println("Program ends ");

    }

    public int divide(int a, int b) throws SoteException {
        System.out.println("We are in myMethod");

        if (b == 0) {
            throw new SoteException("C:\\fake");
        }

        return a / b;
    }
}
```

```java
package errors;

public class StartUsingOurException {
    public static void main(String[] args) {

        StartUsingOurException instance = new StartUsingOurException();

        try {
            instance.divide(2, 3);
        } catch (SoteException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("Program ends ");
        }
        System.out.println("Program ends ");

    }

    public int divide(int a, int b) throws SoteException {
        System.out.println("We are in myMethod");

        if (b == 0) {
            throw new SoteException("C:\\fake");
        }

        return a / b;
    }
}
```

finally blocks are always(!) executed. Typical use case: Working with resources such as databases, networks, filesystems and the resource has to be released.