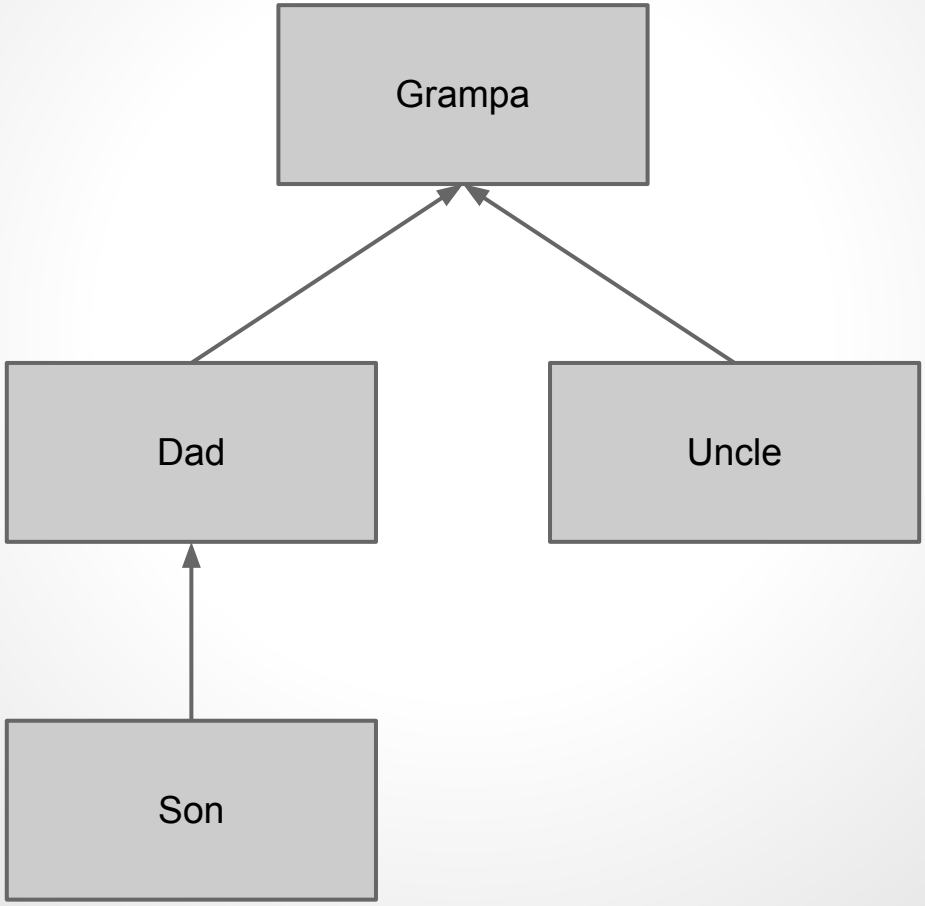
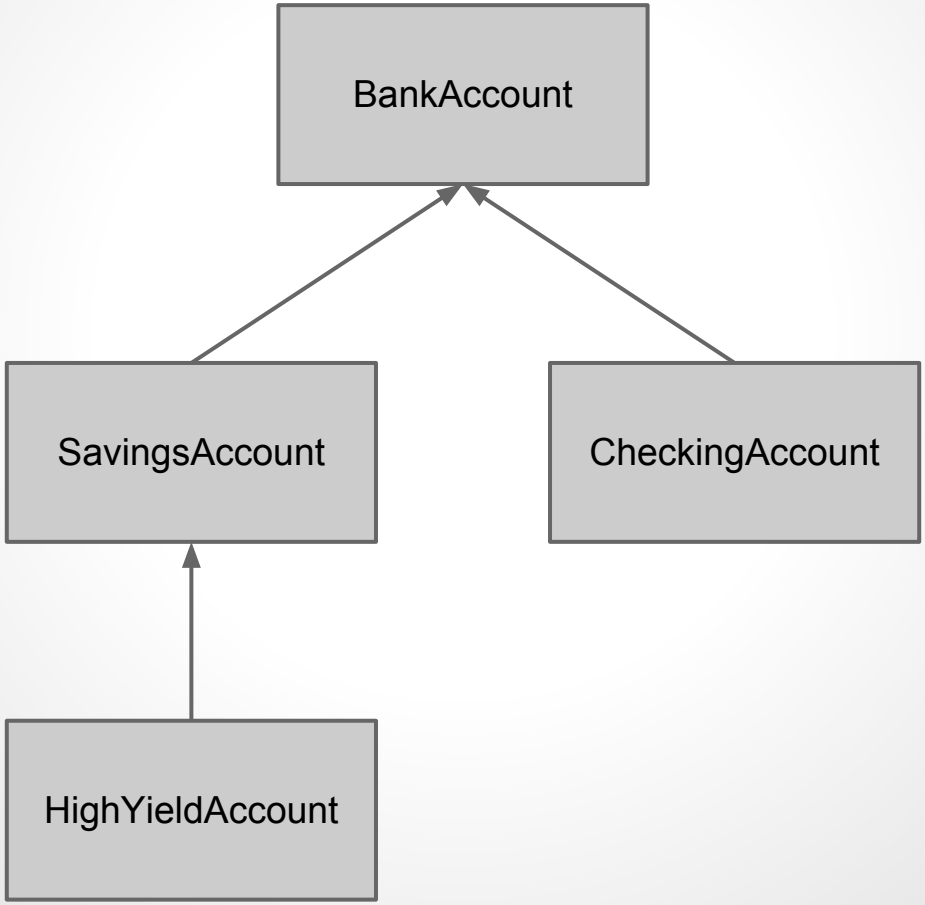
Intermediate Java

Inheritance: Who's Your Daddy?

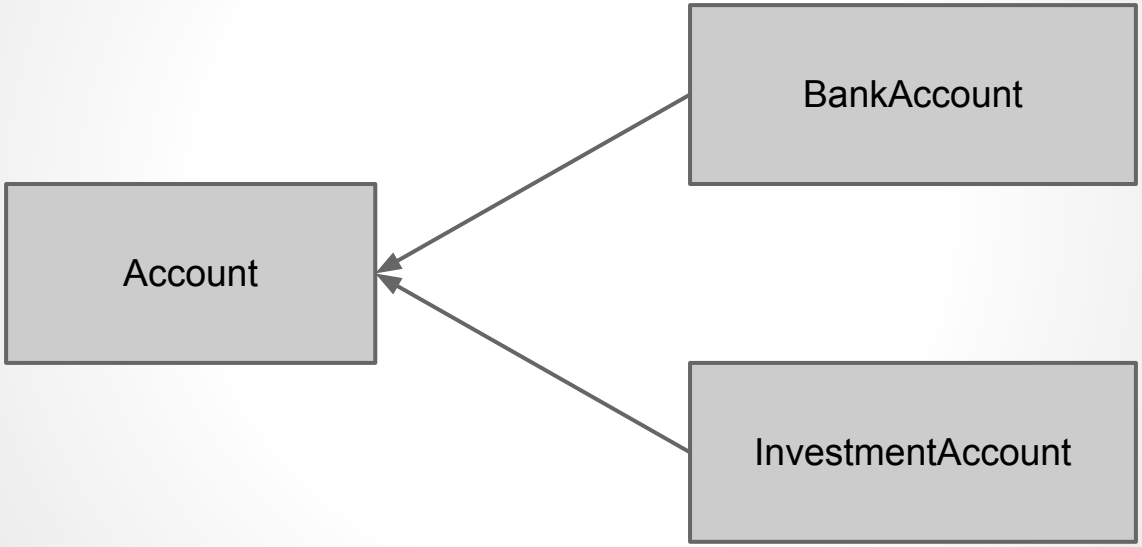


Inheritance



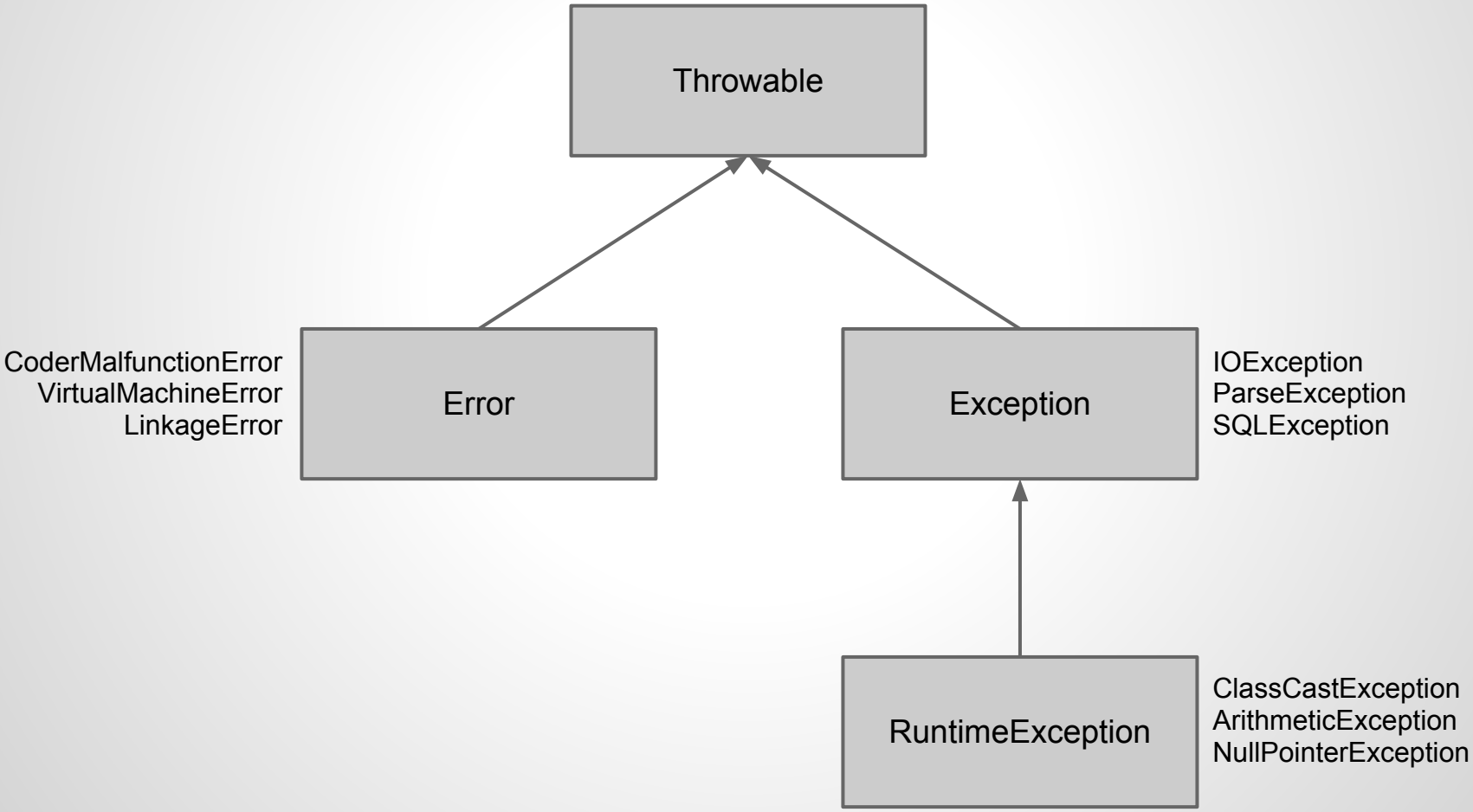
Inheritance Demo

Interfaces



Interface Demo

Exceptions


Throwable
Error
Exception
RuntimeException
CoderMalfunctionError
VirtualMachineError
LinkageError
IOException
ParseException
SQLException
ClassCastException
ArithmeticException
NullPointerException

Defining Your Own Exceptions

```
public class MyException extends Exception {

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

     MyException(String message, Throwable cause) {
          super(message, cause);
     }

}
```

Throwing & Catching Exceptions

```
public class SomeClass {

     public void someMethod() {
          try {
               failingMethod();
          } catch (MyException ex) {
               System.out.println(ex.getMessage());  // prints Die!
          }
     }

     private void failingMethod() throws MyException {
          deeperFail();
     }

     private void deeperFail() throws MyException {
          throw new MyExection("Die!");
     }

}
```

Defining Runtime Exceptions

```
public class MyRuntimeException extends RuntimeException {

     MyRuntimeException(String message) {
          super(message);
     }

     MyRuntimeException(String message, Throwable cause) {
          super(message, cause);
     }

}
```

Throwing & Catching RuntimeExceptions

```
public class SomeClass {

     public void someMethod() {
          try {
               failingMethod();
          } catch (MyRuntimeException ex) {
               System.out.println(ex.getMessage());  // prints Die!
          }
     }

     private void failingMethod() {
          deeperFail();
     }

     private void deeperFail() {
          throw new MyRuntimeExection("Die!");
     }

}
```

Runtime vs. Checked Exceptions

Runtime Exceptions are preferred

- there is little to be done for most errors
  - Database, File IO, Network IO
- checked exceptions just clutter the code

Use Checked Exceptions for things that can actually be handled and still provide meaningful functionality to the user.

Use Runtime Exceptions for things that should just cause the application to fail.

Collections

Lists

...are like an array.

...but unbounded.

...ordered.

```
List<String> list = new ArrayList<String>();

String fee = "fee";
String fi = "fi";
String fo = "fo";
String fum = "fum";

list.add(fee);
list.add(fi);
list.add(fo);
list.add(fum);

list.get(0);       // returns 'fee'
list.size();       // returns 4
list.isEmpty();    // return false
```

Sets

...unordered.

...contains only unique items.

```
Set<String> set = new HashSet<String>();

String fee = "fee";
String fi = "fi";
String fo = "fo";
String fum = "fum";

set.add(fee);
set.add(fi);
set.add(fo);
set.add(fum);

set.contains(fee);      // returns true
set.contains("fee");    // returns false
set.size();             // returns 4
set.isEmpty();          // return false
```

Maps

...unordered.

...contains
key/value pairs.

```
Map<String, String> map =
     new HashMap<String, String>();

String fee = "fee";
String fi = "fi";
String fo = "fo";
String fum = "fum";

map.put("FEE", fee);
map.put("FI", fi);
map.put("FO", fo);
map.put("FUM", fum);

map.get("FEE");         // returns 'fee'
map.get("1234");        // returns null
map.size();             // returns 4
map.isEmpty();          // return false
```

For Each

```
List<String> list = new ArrayList<String>();
list.add("Fee");
list.add("Fi");
list.add("Fo");
list.add("Fum");

for (String s : list) {
  System.out.println(s);
}
```

Output

```
Fee
Fi
Fo
Fum
```

Static & Final

Static

One and only one instance of this type

```
public class StaticExample {

  private static String value = "";

  public void setStatic(String value) {
    this.value = value;
  }

  public String getStatic() {
    return this.value;
  }

}

StaticExample a = new StaticExample();
StaticExample b = new StaticExample();

a.setStatic("set from a");
b.getStatic();   // returns set from a
```

Final

Value cannot be changed

```
public class FinalExample {

  public void method(final String s) {
    s = "foo";  // fails
  }

}
```

Static & Final

Combined, they make great constants

```
public class Coins {

  public static final String NICKEL = "nickel";
  public static final String DIME = "dime";
  public static final String QUARTER = "quarter";

}
```