Unit 2: Date/Time Utilities, Formatting, UML Class Diagrams

APPLICATION PROGRAMMING IN JAVA

Principles of Application Development

Four Views of App Development

- Incremental
- Task-oriented
- Test-first
- Continuous integration

Incremental

- Classes/APIs are coded and tested in chunks
  - After writing a significant chunk of code, stop and test it
  - The meaning of *significant* depends on your skill level
  - Almost always test your code after finishing a new class
  - If you find the chunk difficult to test, consider re-writing it
- After completing a chunk of code archive it
  - *Archive* is usually a source code management system
  - Archived code should be thoroughly tested
  - Don’t forget to archive your test code

Task-oriented

- A well-designed project is *chunkable*
  - Different chunks can be assigned to different groups
  - Supports incremental development
- Chunks should be considered deliverable

Test-first Development

- Create a class with method stubs
- Write tests for all the methods; observe that they execute and fail
- Write the code for you class and test it
  - Consider a method immediately after you write it

Continuous Integration

- Write stubs for the methods that glue chunks together
- As you substitute code for stubs, integrate it into the system or application
- Ideally, the application can be tested every day (or more often)

Date and Time Facilities

Introduction

- java.util.Date is all but dead
- java.util.Date is superseded by java.util.Calendar
- Consider using *org.apache.commons.lang3.time*
- Java 8 has new facilities based on ISO 8601; see the *java.time* packages

java.util.time

- LocalDate
  - Encapsulates year-month-day in some locale
  - Does not contain time data
- LocalTime
  - Encapsulates hour-minute-second-nanosecond
  - No date or time-zone data
- LocalDateTime
  - A coupling of LocalDate and LocalTime
  - No time-zone data

Continued on next slide

Continued from previous slide

java.util.time

- OffsetDateTime
  - Adds GMT offset to LocalDateTime
  - Does not provide comprehensive time-zone support
- OffsetTime
  - Adds GMT offset to LocalTime
  - Does not provide comprehensive time-zone support
- ZonedDateTime
  - Adds full time-zone support to LocalDateTime
- ZoneId
  - Encapsulates a time-zone such as Europe/Berlin

Continued on next slide

Continued from previous slide

java.util.time

- ZoneOffset
  - Encapsulates an offset from GMT
- Instant
  - Encapsulates an instant, with nanosecond precision
- Year
  - Encapsulates a year, such as 2012
- Month
  - Enumeration of the months of the year
- YearMonth
  - Encapsulates a month and year, such as *20 October*

Package java.time.chrono

- Generic API for coupling with non-ISO calendars
- Not of great interest to us
- interface ChronoLocalDate
  - Implemented by LocalDate
- interface ChronoLocalDateTime
  - Implemented by LocalDateTime

LocalDate

- static LocalDate now()
  - Obtains the current date
- Month getMonth()
  - Returns the stored month
- int getYear()
  - Returns the stored year
- boolean isAfter(ChronoLocalDate other)
- boolean isBefore(ChronoLocalDate other)
- static LocalDate of(int year, Month month, int day)

LocalDate: Other Methods

- plusDays(long)
- plusMonths(long)
- plusWeeks(long)
- plusYears(long)

- minusDays(long)
- minusMonths(long)
- minusWeeks(long)
- minusYears(long)

- getDayOfMonth()
- getDayOfWeek()
- getDayOfMonth()
- getDayOfYear()
- getDayOfMonth()

- lengthOfMonth()
- lengthOfYear()

LocalDate Demo

```
private static final LocalDate  DUE_DATE      =
    LocalDate.of( 2018, Month.APRIL, 15 );
private static final LocalDate   REWARD_DATE  =
    DUE_DATE.minusMonths( 1 );
private static final LocalDate   PENALTY_DATE =
    DUE_DATE.plusMonths( 1 );
private static final BigDecimal TEN_PERCENT =
    new BigDecimal( ".1" );

public static void main(String[] args)
{
    BigDecimal  amount      = new BigDecimal( "123.85" );
    BigDecimal  adjAmount   = adjustAmount( amount );
    System.out.println( adjAmount );
}
...
```

Continued on next slide

Continued from previous slide

LocalDate Demo

```
private static
BigDecimal adjustAmount( BigDecimal amount )
{
    BigDecimal  adjAmount   = amount;
    LocalDate   today       = LocalDate.now();

    if ( today.isBefore( REWARD_DATE ) )
    {
        BigDecimal reward = amount.multiply( TEN_PERCENT );
        adjAmount = adjAmount.subtract( reward );
    }
    else if ( today.isAfter( PENALTY_DATE ) )
    {
        BigDecimal penalty = amount.multiply(TEN_PERCENT );
        adjAmount = adjAmount.add( penalty );
    }
    else
    ...
```

Formatters

java.util.Formatter

- Inspired by C's printf function
- Uses a variable-length arg list
- Principal method:

```
public Formatter
format( String fmt, Object args … )
```

- *fmt* consists of text plus format specifiers
- Format specifiers begin with %
- Implements the builder pattern

Formatter Constructors

- Generally constructed with a destination
- Common destinations:
  - StringBuilder object
  - File
  - OuputStream
- Must be closed when no longer needed.

Formatter Demo 1

```
public static void main(String[] args)
{
    int             count   = 5;
    LocalDate       date    = LocalDate.now();
    StringBuilder   bldr    = new StringBuilder();

    Formatter   formatter   = new Formatter( bldr );
    String      fmt         =
        "As of %s you have a total of %d followers";
    formatter.format( fmt, date, count );
    System.out.println( bldr );
    formatter.close();
}
```

Don't forget to close your formatter

Output:
As of 2018-02-13 you have a total of 5 followers

Format Specifiers

%[index$][flags][width][.precision]conversion

- index
  - arg specifier applies to
- flag
  - modifies output format
- width
  - determines the width of the output field
- precision
  - further bounds the output field
- conversion

A Few Conversion Specifiers

Specifier	Description
b or B	Use with Boolean arg
S or S	Use with Object arg
c or C	Use with Unicode character (char)
d	Use with integer arg; decimal output
o	Use with integer arg; octal output
x	Use with integer arg; hexadecimal output
e or E	Use with floating point arg; scientific notation output
f	Use with floating point arg; decimal output
t	Use with date/time arg; see date/time conversions
%	Literal percent sign
n	Platform-dependent line separator

Examples

Given:

```
StringBuilder  bldr = newStringBuilder();
Formatter formatter = new Formatter( bldr );

int          iNum    = 0;
double       fNum    = 0;
double       dNum    = 0;
String       fmt     = null;
LocalDate    date    = null;
String       str1    = null;
String       str2    = null;
```

Conversion Specifier Example

```
iNum = 32;
fNum = .5;
dNum = iNum * fNum;
fmt  = "%d * %f = %f";
formatter.format( fmt, iNum, fNum, dNum );
System.out.println( bldr );
```

Output:

```
32 * 0.500000 = 16.000000
```

Continued on next slide

Continued from previous slide

Conversion Specifier Example

```
iNum = 32;
fNum = .5;
dNum = iNum * fNum;
fmt  = "The product of %d and %e is %e";
formatter.format( fmt, iNum, fNum, dNum );
System.out.println( bldr );
```

Output:

```
The product of 32 and 5.000000e-01 is 1.600000e+01
```

Continued from previous slide

Conversion Specifier Example

```
str1 = "Fred";
str2 = "Wilma";
date = LocalDate.now().plusDays( 1 );
fmt  = "%s and %s are meeting tomorrow, %s";
formatter.format( fmt, str1, str2, date );
System.out.println( bldr );
```

Output:

```
Fred and Wilma are meeting tomorrow, 2018-02-14
```

Width Example 2

```
fmt  = "%3d * %3d = %4d%n";
formatter.format( fmt, 21, 7, 21 * 7 );
formatter.format( fmt, 122, 4, 122 * 4 );
formatter.format( fmt, 5, 202, 5 * 202 );
```

Result

```
 21 *   7 =  147
122 *   4 =  488
  5 * 202 = 1010
```

Output is right-justified in field

Width Example

```
fmt  = "%7s and %7s are %2d years old%n";
formatter.format( fmt, "Sam", "Gilly", 7 );
formatter.format( fmt, "Nathan", "Sandy", 22 );
formatter.format( fmt, "Alice", "Ted", 3 );
```

Result

```
    Sam and   Gilly are  7 years old
 Nathan and   Sandy are 22 years old
  Alice and     Ted are  3 years old
```

Output is right-justified in field

Flag Example 1

```
fmt  = "%-7s and %-7s are %2d years old%n";
formatter.format( fmt, "Sam", "Gilly", 7 );
formatter.format( fmt, "Nathan", "Sandy", 22 );
formatter.format( fmt, "Alice", "Ted", 3 );
```

flag: -

Result

```
Sam     and Gilly   are  7 years old
Nathan  and Sandy   are 22 years old
Alice   and Ted     are  3 years old
```

Output is leftt-justified in field

Flag Example 2

```
fmt  = "%s owes %s %,d dollars%n";
formatter.format( fmt, "Ted", "Suzy", 1250 );
fmt  = "%s owes %s %,8d dollars%n";
formatter.format( fmt, "Ted", "Suzy", 1250 );
```

Result

```
Ted owes Suzy are 1,250 dollars
Ted owes Suzy are  1,250 dollars
```

Output contains group separator

More Flags

Flag	Description	Example	Result
+	Numeric result always has a sign	("%+d, %+d", 4, -4)	+4, -4
[space]	Positive numbers preceded by [space]	(">% d< >% d<", 4, -4)	> 4< >-4<
0	Integer fields 0-fill	("Fine: %03d", 4)	004
(	Negative numbers parenthesized	("%(d, %(d", 3, -3)	3, (3)

Precision Example 1

```
formatter.format( "%8.3f%n%8.3f%n", Math.PI, Math.E );
formatter.format( "%08.3f%n%08.3f%n", Math.PI,Math.E );
formatter.format( "%.3f%n%.3f%n", Math.PI, Math.E );
```

Result

```
   3.142
   2.718
0003.142
0002.718
3.142
2.718
```

Index specifier

- Identifies the argument to which a specifier applies

```
StringBuilder   bldr    = new StringBuilder();
Formatter       fmtr    = new Formatter( bldr );

fmtr.format( "%1$d in hex is 0x%1$x", 282 );
System.out.println( bldr );

bldr.delete( 0, bldr.length() );
fmtr.format( "char: %3$c, int %1$d. dbl: %2$f",
                  29, .5, 'X' );
System.out.println( bldr );

fmtr.close();
```

Formatter Shortcuts

- String.format(String, Object…)

```
String demo = String.format( "%d%s", 3, " French hens" );
```

- PrintStream.printf(String, Object…)

```
System.out.printf( "Call me %s%n", "Ishmael" );
```

Time Conversions

- All specifiers preceded by *t*, for example:

```
fmtr.format( "Hour = %tH", time )
```

Specifier	Description
H	Hour in range 1 to 24
I (“eye”)	Hour in range 1 to 12
k	Hour in range 0 to 23
l (“ell”)	Hour in range 0 to 11
M	Minute in hour, 00 to 59
S	Second in minute, 00 to 59
L	Millisecond within second, 000 to 999
Z	Time zone, adjusted for Daylight Saving Time
p	am/pm

Date Conversions

- All specifiers preceded by *t*, for example:

```
fmtr.format( "Day = %td", time );
```

Specifier	Description
B	Full month name
b	Abbreviated month name
A	Full name of day
a	Abbreviated name of day
C	Two-digit year, 00 to 99
Y	Four-digit year, 0000 to 9999+
m	Month, 00 – 13
d	Day of month, 00 – 31
e	Day of month, 0 – 31

Date/Time Conversions, Examples

- Given:

```
LocalDateTime  time    = LocalDateTime.now();
StringBuilder  bldr    = new StringBuilder();
Formatter      fmtr    = new Formatter( bldr );
```

Date/Time Conversions, Examples

```
fmtr.format( "%1$te-%1$tB-%1$tY", time );
       14-February-2018

fmtr.format( "The time is: %1$tH:%1$tM%1$tp", time );
       The time is: 12:12pm
```

DateTimeFormatter

- Formats a date as a string
- Parses a date from a string
- Alphabetic characters reserved for use in patterns
- Number of pattern characters may determine width of output field
- Many predefined formatters

Pattern Characters

Char	Example	Result
M (month)	“M”	2
	“MM”	02
	“MMM”	Feb
	“MMMM”	February
d (day of month)	“d”	16
u (year)	“u”	2018
	“uu”	18
	“uuu”	2018
z (time-zone)	“z”	PST

More Common Pattern Characters

Char	Description
Q	Quarter of year
E	Day of week (Monday through Sunday)
a	AM/PM
h	Hour (0 through 23)
K	Hour (0 through 11)
k	Hour (1 through 24)
H	Hour (0 through 23)
m	Minute-of-hour (0 through 59)
s	Second of minute (0 through 59

DateTimeFormatter Examples

- Given:

```
ZonedDateTime        time    = ZonedDateTime.now();
DateTimeFormatter    fmtr    = null;
String               str     = null;
```

Pattern	Result
hh:mm:ss	11:09:03
E MMM dd, uuuu	Fri Feb 16, 2018
E MMM dd, uuuu KK:mm a	Fri Feb 16, 2018 11:09 AM

DateTimeFormatter Examples

Pattern	Result
`hh:mm:ss`	`11:09:03`
`E MMM dd, uuuu`	`Fri Feb 16, 2018`
`E MMM dd, uuuu KK:mm a`	`Fri Feb 16, 2018 11:09 AM`

Predefined DateTimeFormatters

- ISO_LOCAL_DATE
- ISO_LOCAL_TIME
- ISO_LOCAL_DATE_TIME
- ISO_ZONED_DATE_TIME
- etc.

Predefined DateTimeFormatters, Examples

```
fmtr = DateTimeFormatter.ISO_LOCAL_DATE;
str = fmtr.format( time );
   2018-02-16

fmtr = DateTimeFormatter.ISO_LOCAL_DATE_TIME;
str = fmtr.format( time );
   2018-02-16T11:54:24.699

fmtr = DateTimeFormatter.ISO_LOCAL_DATE;
str = fmtr.format( time );
   2018-02-16T11:54:24.699-08:00[America/Los_Angeles]
```

More Formatters

- See also: *java.text*

Review: Overriding equals(), hashCode()

Overriding equals(Object)

- Typically, two objects are equal if they're the same type, and all their fields are equal
  - There can be exceptions
- Standard pattern:

```
boolean equals( Object obj )
    this == obj ? true
    else obj == null ? false
    else this-class != obj-class? false
    else
    {
        cast obj to same type as this
        compare corresponding fields
    }
```

Overriding equals, Example

```
public class EqualsDemo
{
    private int     iField;
    private String  sField;
    public boolean equals( Object obj )
    {
        boolean rcode   = false;

        if ( this == obj )
            rcode = true;
        else if ( obj == null )
            rcode = false;
        else if ( !(obj instanceof EqualsDemo) )
            rcode = false;
        else
            ...
```

Continued on next slide

Continued from previous slide

Overriding equals, Example

```
    ...
    {
        EqualsDemo  that    = (EqualsDemo)obj;
        if ( this.iField != that.iField )
            rcode = false;
        else if ( this.sField == that.sField )
            rcode = true;
        else if ( this.sField == null )
            rcode = false;
        else
            rcode = this.sField.equals(that.sField);
    }

    return rcode;
}
```

Objects.equals(Object, Object)

- Since Java 7
- Both objects null? true
- Else first object null? false
- Else first.equals(second)

Objects.equals, example

```
public boolean equals( Object obj )
{
    boolean rcode   = false;

    if ( this == obj )
        rcode = true;
    else if ( obj == null )
        rcode = false;
    else if ( !(obj instanceof ObjectsEqualsDemo) )
        rcode = false;
    ...
```

Continued on next slide

Continued from previous slide

Objects.equals, example

```
    ...
    else
    {
        ObjectsEqualsDemo that =
            (ObjectsEqualsDemo)obj;
        if ( iField != that.iField )
            rcode = false;
        else
            rcode =
            Objects.equals( sField,  that.sField );
    }

    return rcode;
}
```

Overriding hashCode()

- When you override equals() you MUST override hashCode
- See also:
  - Objects.hash()
  - Object.hashCode()
- For a comprehensive discussion of hash codes, see *Effective Java*, by Josh Bloch

```
public int hashCode()
{
    int hash    = iField;
    if ( sField != null )
        hash ^= sField.hashCode();
    return hash;
}
```

Bloch's Hashing Algorithm*

- For fields *1* to *n* compute hash values *1* to *m*

Type	Computation Algorithm
boolean	$field_i$? 1231 : 1237
byte, short, char, int	(int) $field_i$
long	(int) ($field_i$ ^ ($field_i$ >>> 32))
float	Float.floatToIntBits($field_i$)
double	Double.doubleToLongBits($field_i$); apply long algorithm (above)
Object	null ? 0 : $field_i$.hashCode()
Array	Produce a separate hash value for each element of the array

Continued on next slide

*From Bloch, Joshua. *Effective Java*. Addison-Wesley, 2018

Continued from previous slide

Bloch's Hashing Algorithm*

```
result = 1
for i = 1 to m
    result = result * 31 + value_i
```

*From Bloch, Joshua. *Effective Java*. Addison-Wesley, 2018

Objects.hashCode(Object)

- Given:

```
public class ObjectsHashCodeDemo
{
    private int     iVar    = 37;
    private float   fVar    = 3.14159f;
    private double  dVar    = 2.71828;
    private String  sVar    = "Sally";
    private boolean bVar    = true;

    private int[]   iArr    = { -1, 2, 4, -7 };
```

Continued on next slide

Continued from previous slide

Objects.hashCode(Object)

```
public void test()
{
    System.out.println( Objects.hashCode( iVar ) );
    System.out.println( Objects.hashCode( fVar ) );
    System.out.println( Objects.hashCode( dVar ) );
    System.out.println( Objects.hashCode( sVar ) );
    System.out.println( Objects.hashCode( bVar ) );
    System.out.println( Objects.hashCode( null ) );
}
```

```
37
1078530000
-709932903
79649227
1231
0
```

Objects.hash(Object...)

- Derives hash code from *n* objects

```
public void test()
{
    int hash    =
        Objects.hash( iVar, fVar, dVar, sVar, bVar, iArr );
    System.out.println( hash );
}
```

JUnit Test for equals/hashCode: testEqualsHash

```
@Test
public void testEqualsHash()
{
    int     iVal1   = 10;
    int     iVal2   = -iVal1;
    String  str1    = "Camel";
    String  str2    = str1 + "Alpaca";

    EqualsDemo2 demo         = new EqualsDemo2(iVal1, str1);
    EqualsDemo2 equal        = new EqualsDemo2(iVal1, str1);
    EqualsDemo2 notEqual     = new EqualsDemo2(iVal2, str2);
    EqualsDemo2 notEqualInt  = new EqualsDemo2(iVal2, str1);
    EqualsDemo2 notEqualStr  = new EqualsDemo2(iVal2,str1 );
    ...
```

Continued on next slide

JUnit Test for equals/hashCode: testEqualsHash

Continued from previous slide

```
    ...
    assertEquals( demo, demo );
    assertEquals( demo, equal );
    assertEquals( equal, demo );
    assertEquals( demo.hashCode(), equal.hashCode() );

    assertNotEquals( demo, null );
    assertNotEquals( demo, new Object() );
    assertNotEquals( demo, notEqual );
    assertNotEquals( notEqual, demo );
    assertNotEquals( demo, notEqualInt );
    assertNotEquals( notEqualInt, demo );
    assertNotEquals( demo, notEqualStr );
    assertNotEquals( notEqualStr, demo );
}
```

JUnit Test for equals/hashCode: testEqualsHashNull

```
public void testEqualsHashNull()
{
    int     iVal    = 10;
    String  sVal    = "Llama";

    EqualsDemo2 nonNullStr  = new EqualsDemo2(iVal, sVal);
    EqualsDemo2 nullStr1    = new EqualsDemo2(iVal, null);
    EqualsDemo2 nullStr2    = new EqualsDemo2(iVal, null);

    assertEquals(nullStr1, nullStr2 );
    assertEquals(nullStr1.hashCode(), nullStr2.hashCode());

    assertNotEquals(nonNullStr, nullStr1);
    assertNotEquals(nullStr1, nonNullStr);
}
```

Class Diagrams

UML

UML Class Diagrams, Perspectives

- Three ways to view class diagrams:
  - Conceptual perspective
  - Specification perspective
  - Implementation perspective

UML Class Diagrams, Perspectives

- Conceptual
  - Diagrams represent conceptual strategy
  - No mapping to “real” classes
- Specification
  - Diagrams represent interfaces
  - Encapsulate high-level details
  - Usually a weak mapping to actual implementation
  - Most useful perspective
- Implementation
  - Diagrams represent details of concrete implementation

UML Class Diagrams, Conceptual Perspective

- Diagrams represent conceptual strategy
- No mapping to “real” classes
- May be of little use in later phases of development

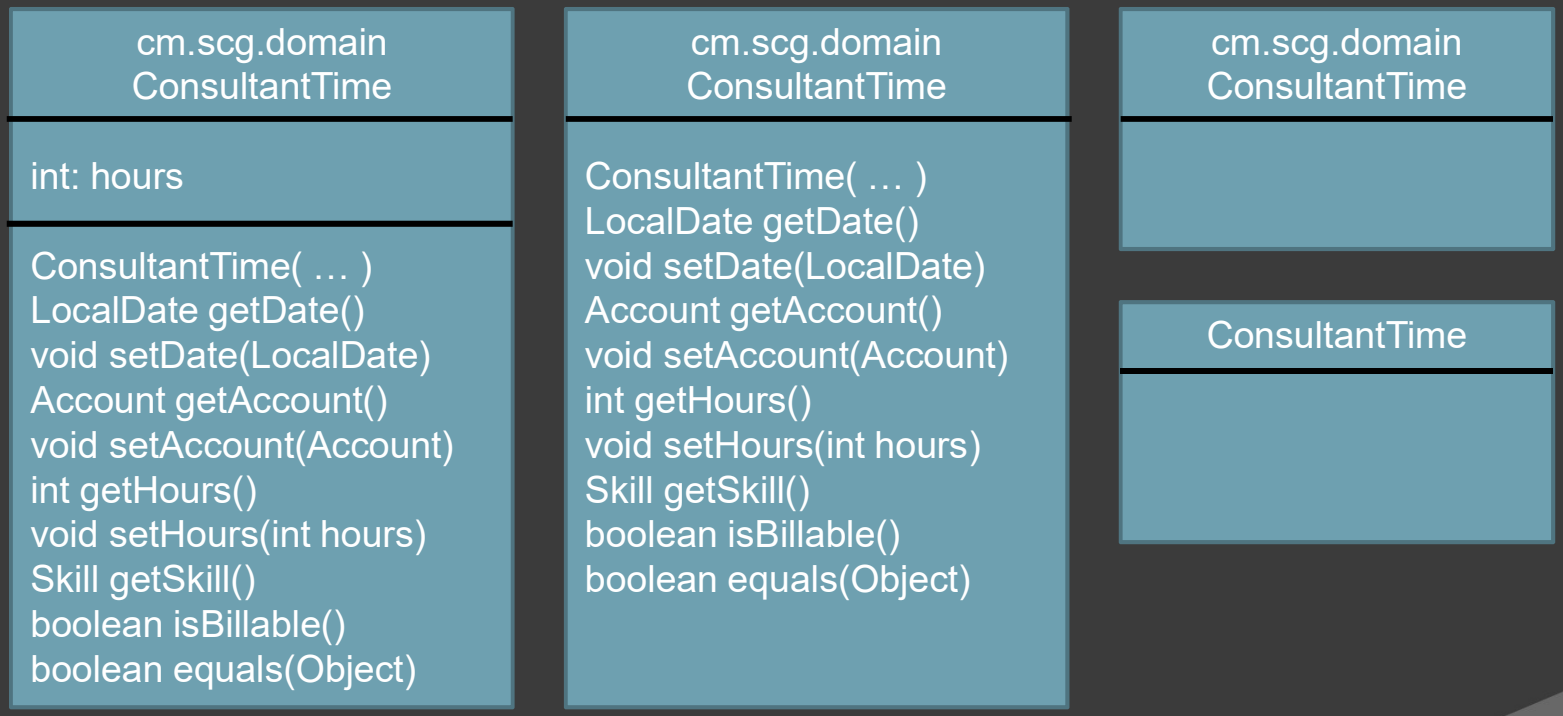
UML Class Diagrams, Specification Perspective

- Diagrams represent interfaces
- Capture high-level details
- Usually a weak mapping to actual implementation
- Most useful perspective

UML Class Diagrams, Implementation Perspective

- Diagrams encapsulate details of concrete implementation
- Can be difficult to read
- Probably should NOT encapsulate every method of every class, or even every class

UML Class Representation, Example 1



Associations

Contains

Extends

Implements

Associations, Examples