Date and time manipulation

Topics

- RPG date and time review
- Java date and time preview
- Getting a date
- Comparing dates
- Calculating duration
- Extracting date and time parts
- Formatting date and time
- Leap year calculations
- Time zones

Unit objectives

After completing this unit, you should be able to:

- Encapsulate the current date and time using classes in the Java Development Kit (JDK)
- Manipulate and compare date and time values in Java
- Display date and time values to specific formatting patterns and locales

RPG III date support

- Pre-V2R2:
  - Keywords:
    - UYEAR, UMONTH, UDAY, UDATE
  - TIME op-code
    - 6- or 12-digit result field
    - hhmmss or hhmmssDDDDDD

- V2R2:
  - New keywords:
    - *DATE, *YEAR, *MONTH, *DAY
  - TIME op-code
    - 6-, 12- or 14-digit result field
    - 4-digit year

RPG IV date support

- New data types
  - Date ('D')
    - Holds date, formatted according to DATFMT keyword
  - Time ('T')
    - Holds time, formatted according to TIMFMT keyword
  - Timestamp ('Z')
    - Holds date and time, formatted as: yyyy-mm-dd-hh.mm.ss.mmmmmm
- New and updated op-codes
  - ADDDUR, SUBDUR, EXTRCT, TEST, MOVE

Java date and time

Class	**Pack-age**	**Description** Java
Date	`java.util`	Simple date and time capture; no manipulation methods
Gregorian-Calendar	`java.util`	Rich date-time functionality, including comparing, adding, subtracting, and extracting
SimpleDate-Format	`java.text`	For creating "formatting objects" that will format any given Date object to the specified format pattern
TimeZone	`java.util`	For creating TimeZone objects representing any time zone. Apply to GregorianCalendar or SimpleDateFormat objects to get equivalent date and time in that time zone

Current date and time

RPG

```
C                   GETDATTME     BEGSR
C                                 TIME                    TM      14 0
C                                 ENDSR
```

14-digit date + time

Java

```
import java.util.*; // for Date class

public class MyTime
{
    public static void main(String args[])
    {
        Date today = new Date();
        System.out.println("Time is: " + today);
    }
}
```

Raw date. No manipulation allowed

formats Date object

Time is: Wed Feb 18 11:29:45 EST 2004

Java

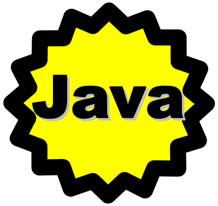
```
import java.util.*; // for GregorianCalendar class
...

        GregorianCalendar today = new GregorianCalendar();
        System.out.println("Time is: " + today.getTime());
```

Smart Date Manipulation supported

returns Date object

CurrentTimeMillis

- Another option is currentTimeMillis
  - Static method in java.lang.System class
  - Returns raw number of milliseconds since EPOCH
    - EPOCH in Java = 1 January 1970
    - As a long value
  - Useful for measuring elapsed time



```
public class MyTime
{
    public static void main(String args[])
    {
        long time1 = System.currentTimeMillis();
        long time2 = System.currentTimeMillis();
        System.out.println("Elapsed time: " +
                           time2 - time1);
    }
}
```

Date class

- Date class represents a specific instant in time, to millisecond precision.
  - Used to also support formatting and parsing
    - But this function is now handled by GregorianCalendar
- Methods
  - Before, after, equals (Date when) "deprecated"
    - Compares this date to another date; returns true or false
  - getTime() / setTime(long)
    - Retrieves or sets the number of milliseconds since 1 January 1970
  - toString()
    - Returns formatted string of form:
      - dow month dd hh:mm:ss zzz yyyy
        - > Where dow = day of week (Sun, Mon, ...)

GregorianCalendar (1 of 5)

- GregorianCalendar class represents calendar used in most of world.
  - But Java's design allows for other calendar classes in the future
- Knows:
  - How to correctly add or subtract, correctly "rolling"
  - How to compare two dates
  - About leap years (isLeapYear method)
  - About eras (BC and AD)

GregorianCalendar (2 of 5)

- GregorianCalendar has following *constructors* (not a complete list):

Java

Parameters	Description
`none`	Holds current date and time
`(int year, int month, int day)`	Holds given date, zero time. **Note:** month is zero-based; year and day are not.
`(int year, int month, int day, int hour, int minute)`	Holds given date and time. Only the month is zero-based. Hour is 24-hour.
`(int year, int month, int day, int hour, int minute, int sec)`	Holds given date and time, down to the second

```
gc = new GregorianCalendar(2001, 4, 14, 20, 18, 30);
System.out.println("Date and Time:  " + gc.getTime());
```

```
Date and Time:  Wed Feb 18 11:29:45 EST 2004
```

GregorianCalendar (3 of 5)

- GregorianCalendar has these *methods* (not a complete list):

Methods	Description
`add(int field, int amount)`	Adds years, months, days, hours, minutes, or seconds. Use constants for first parameter to identify what the second parameter is. To subtract, pass negative number for second parm.
`after(Object otherDate)`	Returns true if this date is after the given other date.
`before(Object otherDate)`	Returns true if this date is before the given other date.
`equals(Object otherDate)`	Compares this date to another, return true or false.
`get(int field)`	Returns the year, month, day, hour, minute or second (all ints). Can also extract other things. Use constants for parameter to identify which part you want to get. There is also a similar set method for setting these parts.
`getTime()`	Returns date as a Date object. There is also a setDate method to set the date value from a Date object.
`getTimeInMillis()`	Returns a long which is the elapsed time in milliseconds since the epoch. Also there is a set method.
`isLeapYear(int year)`	Returns true if given year is a leap year.

GregorianCalendar (4 of 5)

- GregorianCalendar has following *constants* (not a complete list), which are used in add, get and set methods, and in constructors:

Constants	Description
`ERA, BC, AD`	ERAs. Use ERA on get, which returns BC or AD.
`AM_PM, AM, PM`	Use AM_PM on get, which returns AM or PM.
`YEAR, MONTH`	Use on get to return the year or month as an integer. **Note**: returned month is zero-based.
`DATE, DAY_OF_MONTH, DAY_OF_WEEK, DAY_OF_WEEK_IN_MONTH, DAY_OF_YEAR`	Use to get the day. **Note**: DATE is the same as DAY_OF_MONTH.
`HOUR, MINUTE, SECOND, MILLISECOND`	Use on get to return hour, minute, second or milliseconds.
`WEEK_OF_MONTH, WEEK_OF_YEAR`	Use on get to retrieve the week number.
`JANUARY, FEBRUARY ... DECEMBER`	Constants representing the months.

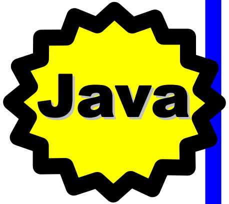
GregorianCalendar (5 of 5)

- Note the following about the constants:
  - They are actually defined in the *parent* class (*parent* is defined in chapter 9) of the GregorianCalendar class.
  - This means you can use either class name to qualify them:
    - Calendar.YEAR or
    - GregorianCalendar.YEAR
  - Use CALENDAR.xxxx

Comparing dates

- RPG: use IF op-code to test:
  - `IF      date1 > date2`
- Java: use before or after method in GregorianCalendar:

```
import java.util.*;
public class TestDate2
{
   public static void main (String args[])
   {
       GregorianCalendar gc1 =
          new GregorianCalendar(1999, 11, 31);
       GregorianCalendar gc2 =
          new GregorianCalendar(2000, 0, 1);
       if (gc1.before(gc2))
         System.out.println("Yes it is");
       else
         System.out.println("No it is not");
   }
}
```



```
Yes it is
```

Duration in RPG

- In RPG IV use: ADDDUR and SUBDUR

```
DstartD           S                D   DATFMT(*ISO) INZ(D'02 18 98')
DendD             S                D   DATFMT(*ISO) INZ(D'02 28 98')
C*
C      endD            SUBDUR    startD         reslt:*D          2 0
```

Date Part	Keyword	Short
Year	*YEARS	*Y
Month	*MONTHS	*M
Day	*DAYS	*D
Hour	*HOURS	*H
Minute	*MINUTES	*MN
Second	*SECONDS	*S
Microsecond	*MSECONDS	*MS

Duration in Java

- Use add method in GregorianCalendar

```
import java.util.*;

public class TestDate
{
  public static void main(String args[])
  {
    GregorianCalendar gc = new GregorianCalendar();
    System.out.println("Date before addition: " + gc.getTime());

    gc.add(Calendar.DATE,2); // add two days
    System.out.println("Date after addition:  " + gc.getTime());

    gc.add(Calendar.YEAR,2);
    System.out.println("Date after addition:  " + gc.getTime());
   }
}
```

Use negative number to subtract.

```
Date before addition: Tue Jun 20 21:14:55 EDT 2000
Date after addition:  Thu Jun 22 21:14:55 EDT 2000
Date after addition:  Sat Jun 22 21:14:55 EDT 2002
```

Rolling dates

- What if date is end of month and you add a few days to it?

Java

```java
GregorianCalendar gc = new GregorianCalendar(2000,10,30);
System.out.println("The date before addition: " + gc.getTime());
gc.add(Calendar.DATE,2);
System.out.println("The date after addition:  " + gc.getTime());
gc.add(Calendar.MONTH,2);
System.out.println("The date after addition:  " + gc.getTime());
gc.add(Calendar.DATE,26);
System.out.println("The date after addition:  " + gc.getTime());
gc.add(Calendar.DATE,1);
System.out.println("The date after addition:  " + gc.getTime());
```

```
The date before addition: Thu Nov 30 00:00:00 EST 2000
The date after addition:  Sat Dec 02 00:00:00 EST 2000
The date after addition:  Fri Feb 02 00:00:00 EST 2001
The date after addition:  Wed Feb 28 00:00:00 EST 2001
The date after addition:  Thu Mar 01 00:00:00 EST 2001
```

Identifying date part

- What part of a date are you adding?

Date Part	RPG	JAVA
Year	`*YEARS or *Y`	`calendar.YEAR`
Month	`*MONTHS or *M`	`Calendar.MONTH`
Day	`*DAYS or *D`	`Calendar.DATE`
Hour	`*HOURS or *H`	`Calendar.HOUR`
Minute	`*MINUTES or *MN`	`Calendar.MINUTE`
Second	`*SECONDS or *S`	`Calendar.SECOND`
Microsecond	`*MSECONDS or *MS`	`Calendar.MILLISECOND`

Day of week: RPG

```
D Week            S               9A   DIM(7) CTDATA PERRCD(1)
D CurrentDate     S                D
D OneSunday       S                D   INZ(D'1999-05-10')
D Temp            S               7  0
D TheDay          S               1P 0
C      'enter date'  DSPLY                       CurrentDate
C      CurrentDate   SubDur    OneSunday         Temp:*d
C      Temp          DIV       7                 Temp
C                    MVR                         TheDay
C                    IF        TheDay = 0
C                    EVAL      TheDay = 7
C                    ENDIF
C      Week(TheDay)  DSPLY
C                    MOVE      *ON               *INLR
** CTDATA Week
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

Day of the week: Option 1

- First option: use GregorianCalendar.get()...

```
import java.util.*;
class MyDayString
{
  public static void main(String args[])
   {
     String WeekDay[] = {"Sunday",    "Monday",   "Tuesday",
                         "Wednesday", "Thursday","Friday",
                         "Saturday"};
     GregorianCalendar dateToday = new GregorianCalendar();
     System.out.println("Today :" +
           WeekDay[dateToday.get(Calendar.DAY_OF_WEEK)]);
   }
}
```

- A better option is coming...

Extracting date parts: RPG

- RPG IV: use EXTRCT op-code
- Use with D, T, or Z fields. Returns one:
  - Year, month, or day
  - Hours, minutes, or seconds
  - Microseconds

```
D CurrentDate     S               D
D OneSunday       S               D   INZ(D'2000-06-25')
D thistime        S               T   INZ(T'11.25.00')
D TStamp          S               Z
C                   MOVE      OneSunday     TStamp
C                   MOVE      thistime      TStamp
C                   EXTRCT    TStamp:*H     temp1             4 0
C     temp1         DSPLY
C                   EXTRCT    OneSunday:*M  temp1
C     temp1         DSPLY
C                   EXTRCT    OneSunday:*Y  temp1
C     temp1         DSPLY
C                   MOVE      *ON           *INLR
```

Extracting date parts: Java

- Java: use get method of GregorianCalendar class
  - Use appropriate constant Calendar.XXX:
    - Calendar.YEAR, Calendar.MONTH, Calendar.DAY
    - Calendar.HOUR, Calendar.HOUR_OF_DAY, Calendar.MINUTE, Calendar.SECOND, Calendar.MILLISECOND

```
import java.util.*;
public class MyDateParts
{
   public static void main(String args[])
   {
       GregorianCalendar date = new GregorianCalendar();
       int temp1 = date.get(Calendar.HOUR);
       System.out.println("Hour = " + temp1);
       temp1 = date.get(Calendar.HOUR_OF_DAY);
       System.out.println("24Hour = " + temp1);
       temp1 = date.get(Calendar.MONTH);
       System.out.println("Month = " + temp1);
       temp1 = date.get(Calendar.YEAR);
       System.out.println("Year = " + temp1);
   }
}
```

Java

```
Hour = 2
24Hour = 14
Month = 6
Year = 2000
```

Leap year

- What is the correct leap year algorithm?
- Is the year a multiple of 400?
  - If yes, it is a leap year. Just skip the following two steps
- If not, then is the year a multiple of a 100?
  - If yes, then it is not a leap year. Skip the next step
- Otherwise is the year a multiple of 4?
  - If yes, then it is a leap year

Leap year in Java

- Java supplies a method to query "leap year-ness"...

```java
import java.util.*; // for GregorianCalendar class

public class TestLeap
{
  public static void main(String args[])
   {
    GregorianCalendar date = new GregorianCalendar();
    System.out.println("1600 a leap year? " + date.isLeapYear(1600));
    System.out.println("1900 a leap year? " + date.isLeapYear(1900));
    System.out.println("1976 a leap year? " + date.isLeapYear(1976));
    System.out.println("1999 a leap year? " + date.isLeapYear(1999));
    System.out.println("2000 a leap year? " + date.isLeapYear(2000));
   }
}
```

Java

```
1600 a leap year? True.
1900 a leap year? False.
1976 a leap year? True.
1999 a leap year? False.
2000 a leap year? True.
```

Date versus GregorianCalendar

- In summary, there are two options for dates in Java:
  - The Date class
  - The GregorianCalendar class
- Use Date class when all you need is the current date for comparison or printing.
- Use GregorianCalendar class when you need to do date math or extraction.

Date formatting in RPG

- Use DATFMT(*format{separator}) to format your date fields.
- DATFMT keyword can be specified on the:
  - H-Specification
  - D-Specification

RPG

Date Format	Name	Format	Length	Sep
Month/Day/Year	*MDY	mm/dd/yy	8	/ - . , &
Day/Month/Year	*DMY	dd/mm/yy	8	/ - . , &
Year/Month/Day	*YMD	yy/mm/dd	8	/ - . , &
Julian	*JUL	yy/ddd	6	/ - . , &
USA Standard	*USA	mm/dd/yyyy	10	/
European Standard	*EUR	dd.mm.yyyy	10	.
International Standard Organization	*ISO	yyyy-mm-dd	10	-
Japanese Standard	*JIS	yyyy-mm-dd	10	-

- Example: H DATFMT(*MDY/)

Date formatting rules

- DATFMT(*format{separator}) on H-Spec: global default
- DATFMT(*format{separator}) on D-Spec: overrides default
  – RPG default date format is *ISO.

```
H************************************************************
H    DATFMT(*YMD/)
H************************************************************
D EURDate              S                  D      DATFMT(*EUR)
D MDYDate              S                 8A
D ISODate              S                  D      DATFMT(*ISO)
D DEFDate              S                  D
D************************************************************
C             *MDY        MOVE        EURDate         MDYDate
C                         MOVE        EURDate         ISODate
C                         MOVE        EURDate         DEFDate
C                         MOVE        *ON             *INLR
```

Date formatting in Java (1 of 4)

- Java supplies SimpleDateFormat class in java.text package (Can specify *any* user-defined pattern)...

```
import java.text.*;
import java.util.*;

public class TestDateFormat
{
   public static void main (String args[])
   {
       Date date = new Date();
       System.out.println("Before formatting: " + date);

       String fPattern = new String("MM/dd/yyyy");
       SimpleDateFormat test = new SimpleDateFormat(fPattern);
       String dateString = test.format(date);
       System.out.println("After formatting:  " + dateString);
   }
}
```

```
Before formatting: Wed Feb 18 21:24:52 EST 2004
After formatting:  02/18/2004
```

Date formatting in Java (2 of 4)

- To establish a new format, just change the pattern.
  - The following pattern substitution variables are supported:

Character	Meaning
`G (uppercase)`	Era designator - text
`y (lowercase)`	year - number
`M (uppercase)`	month in year - text and number
`d (lowercase)`	day in month - number
`E (uppercase)`	day in week - text
`D (uppercase)`	day in year - number
`F (uppercase)`	day of week in month - number
`w (lowercase)`	week in year - number
`W (uppercase)`	week in month - number
`'`	escape for text - delimiter
`''`	single quotes around literals

Date formatting in Java (3 of 4)

Example

Java

```
import java.text.*;
import java.util.*;
public class TestDateFormat2
{
   public static void main (String args[])
    {
        String fPatternA = new String("MM.DD.yyyy G 'JAVA4RPG'");
        SimpleDateFormat testA = new SimpleDateFormat(fPatternA);
        String fPatternB = new String("'Day of week:' EEEE");
        SimpleDateFormat testB = new SimpleDateFormat(fPatternB);

        String dateStringA = testA.format( new Date() );
        System.out.println("Formatted date: " + dateStringA);
        String dateStringB = testB.format( new Date() );
        System.out.println("Formatted date: " + dateStringB);
  }
}
```

```
Formatted date: 12.349.2000 AD JAVA4RPG
Formatted date: Day of week: Monday
```

Date formatting in Java (4 of 4)

- A note about SimpleDateFormat substitution characters:
  - If less than four characters (for example, "MM") then the "short form" of the date part is displayed:
    - Depends on part
    - For example, for months is "Jan", "Feb", and so forth
    - For example, for years is 2-digit years
  - If four or more characters (for example "YYYY") then the "long form" of the date part is displayed:
    - Depends on part
    - For example, for months is "January", and so forth
    - For example, for years is 4-digit years

Day of the week: Option 2

- Second option for getting day of week:
  - Use 'E' substitution variable in format string

Java

```
import java.util.*; // for Date class
import java.text.*; // for SimpleDateFormat class
class MyDayString
{
  public static void main(String args[])
  {
     // option 2 for getting weekday...
     SimpleDateFormat dayFormat = new SimpleDateFormat("E");
     String day = dayFormat.format(new Date());
     System.out.println("Today : " + day);
     SimpleDateFormat dayFmt2 = new SimpleDateFormat("EEEE");
     String day2 = dayFmt2.format(new Date());
     System.out.println("Today : " + dayD);
  }
}
```

```
Sat
Saturday
```

Substitution variable 'E' == Day of Week

Time formatting in RPG

- Use TIMFMT(*format{separator}) to format your date fields.
- TIMFMT keyword can be specified on the:
  - H-Specification
  - D-Specification

Time Format	Name	Format	Len	Sep
Hours:minutes:seconds	*HMS	hh:mm:ss	8	:.,&
International Standard Organization	*ISO	hh.mm.ss	8	.
USA Standard	*USA	hh:mm am/pm	8	:
European Standard	*EUR	hh.mm.ss	8	.
Japanese	*JIS	hh:mm:ss	8	:

- Example: **H TIMFMT(*HMS:)**

Time formatting rules

- TIMFMT(*format{separator}) on H-Spec: global default
- TIMFMT(*format{separator}) on D-Spec: overrides default
  - RPG default date format is *ISO.

```
H*************************************************************
H    TIMFMT(*HMS,)
H*************************************************************
D EURTime              S                  D      TIMFMT(*EUR)
D USATime              S                 8A
D JISTime              S                  D      TIMFMT(*JIS)
D DEFTime              S                  D
D*************************************************************
C             *USA        MOVE        EURTime         USATime
C                         MOVE        EURTime         JISTime
C                         MOVE        EURTime         DEFTime
C                         MOVE        *ON             *INLR
```

Time formatting in Java (1 of 2)

- To establish a new format, just change the pattern.
  - The following pattern substitution variables are supported

Character	Meaning
h	hour in am or pm - number
H	hour within the day - number
m	minute within the hour - number
s	second within the minute - number
S	milliseconds - number
a	am or pm marker - text
k	hour within the day - number
K	hour in am or pm - number
z	time zone - text
'	escape for text - delimiter
''	single quotes around literals

Time formatting in Java (2 of 2)

Example

```
import java.text.*;
import java.util.*;

public class TestTimeFormat2
{
  public static void main(String args[])
  {
     Date date = new Date();
     System.out.println("Before formatting: " + date);
     String fPatt = new String("'Hour in day: ' hh:mm:ss:SS zz");
     SimpleDateFormat test = new SimpleDateFormat(fPatt);
     String timeString = test.format(date);
     System.out.println("After formatting:  " + timeString);
  }
}
```

Java

```
Before formatting: Tue Jun 20 21:33:18 EDT 2000
After formatting:  Hour in day:  09:33:18:43 EDT
```

Default formats (1 of 3)

- To be a good international citizen, you should print or display dates per local formatting conventions and using strings translated to local language.
- Java offers "default formats" for dates and times that will format dates and times that are appropriate for the regional settings of the computer the user is running on:
  - Three static methods in DateFormat class:
    - (SimpleDateFormat)DateFormat.getDateInstance()
    - (SimpleDateFormat)DateFormat.getTimeInstance()
    - (SimpleDateFormat)DateFormat.getDateTimeInstance()

Default formats (2 of 3)

```
import java.text.*;
import java.util.*;

public class TestDefaultFormats
{
    public static void main(String args[])
    {
        Date currentTime = new Date();
        SimpleDateFormat datefmt =
            (SimpleDateFormat)DateFormat.getDateInstance();
        SimpleDateFormat timefmt =
            (SimpleDateFormat)DateFormat.getTimeInstance();
        SimpleDateFormat dttmfmt =
            (SimpleDateFormat)DateFormat.getDateTimeInstance();
        System.out.println(datefmt.format(currentTime));
        System.out.println(timefmt.format(currentTime));
        System.out.println(dttmfmt.format(currentTime));
    }
}
```

Java

```
Feb 18, 2004
12:01:14 PM
Feb 18, 2004 12:01:14 PM
```

Don't want short format? Specify a parameter indicating length of format...

Default formats (3 of 3)

```java
int styles[]     = {DateFormat.SHORT, DateFormat.MEDIUM,
                    DateFormat.LONG,   DateFormat.FULL};
String names[] =  {"SHORT", "MEDIUM", "LONG", "FULL"};
SimpleDateFormat datefmt, timefmt, dttmfmt;

Date currentTime = new Date();
for (int idx = 0; idx < styles.length; idx++)
{
   System.out.println("Style: " + names[idx]);
   datefmt = (SimpleDateFormat)
     DateFormat.getDateInstance(styles[idx]);
   timefmt = (SimpleDateFormat)
     DateFormat.getTimeInstance(styles[idx]);
   dttmfmt = (SimpleDateFormat)
     DateFormat.getDateTimeInstance(
                   styles[idx],styles[idx]);
   System.out.println(datefmt.format(currentTime));
   System.out.println(timefmt.format(currentTime));
   System.out.println(dttmfmt.format(currentTime));
   System.out.println();
}
```

```
Style: SHORT
3/24/04
12:37 AM
3/24/04 12:37 AM

Style: MEDIUM
Mar 24, 2004
12:37:33 AM
Mar 24, 2004 12:37:33 AM

Style: LONG
March 24, 2004
12:37:33 AM CST
March 24, 2004 12:37:33 AM
CST

Style: FULL
Wednesday, March 24, 2004
12:37:33 AM CST
Wednesday, March 24, 2004
12:37:33 AM CST
```

Time zones in Java

- Use TimeZone class in java.util package:

  - `TimeZone tz_GMT = TimeZone.getTimeZone("GMT");`
  - `TimeZone tz_EST = TimeZone.getTimeZone("EST");`

- To see all supported time zone use static method getAvailableIDs as follows:

  -
    ```
    String tzs[] = TimeZone.getAvailableIDs();
    for (int idx = 0; idx < tzs.length; idx++)
      System.out.println(tzs[idx]);
    ```

- Use getTimeZone method to create new TimeZone object; for example, getTimeZone("EST").

TimeZone example

```
import java.util.*;
import java.text.*;
public class TestTimeZones
{
   public static void main(String args[])
   {
    Date     today = new Date(); // Current date and time
    TimeZone tz1   = TimeZone.getTimeZone("PST");
    TimeZone tz2   = TimeZone.getTimeZone("EST");
    TimeZone tz3   = TimeZone.getTimeZone("GMT");
    TimeZone tz4   = TimeZone.getDefault();
    SimpleDateFormat formatter =
      new SimpleDateFormat("hh:mm:ss - 'TZ = ' z");
    formatter.setTimeZone(tz1);
    System.out.println( formatter.format(today) );
    formatter.setTimeZone(tz2);
    System.out.println( formatter.format(today) );
    formatter.setTimeZone(tz3);
    System.out.println( formatter.format(today) );
    formatter.setTimeZone(tz4);
    System.out.println( formatter.format(today) );
   }
}
```

```
06:35:49 - TZ =  PDT
09:35:49 - TZ =  EDT
01:35:49 - TZ =  GMT
09:35:49 - TZ =  EDT
```

Java date and time class summary

- When to use what class?

Class	Usage and Purpose
`java.util.Date`	Basic date and time in milliseconds.
`java.util.Calendar`	Never use directly! Use GregorianCalendar subclass (except for constants defined here).
`java.util.GregorianCalendar`	For converting Date object to "calendar aware" date parts and doing manipulation on it or them.
`java.text.SimpleDateFormat`	For formatting a Date object into a string.
`java.util.TimeZone`	For use with SimpleDateFormat for displaying a date in different time zones.

Java date and time class constructors

- Default constructors per class

Class	Constructor
`java.util.Date`	Current date in milliseconds.
`java.util.Calendar`	No constructor.
`java.util.GregorianCalendar`	Current date. Note: Use `getTime()` method to return as a Date object.
`java.text.SimpleDateFormat`	Given string creates formatter object. No string: default format.
`java.util.TimeZone`	Given string creates TimeZone object. No string: default time zone for your computer.

Topics covered

- RPG date and time review
- Java date and time preview
- Getting a date
- Comparing dates
- Calculating duration
- Extracting date and time parts
- Formatting date and time
- Leap year calculations
- Time zones

Unit summary

Having completed this unit, you should be able to:

- Encapsulate the current date and time using classes in the Java Development Kit (JDK)
- Manipulate and compare date and time values in Java
- Display date and time values to specific formatting patterns and locales