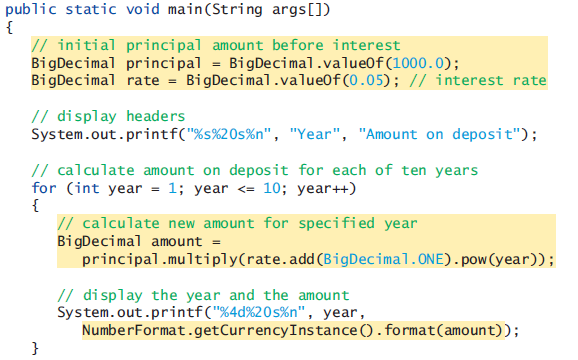
**JAVA API**

* **java.lang**

**-BigDecimal**(java.math.BigDecimal): BigDecimal objects are immutable.



First, the expression rate.add(BigDecimal.ONE) adds 1 to the rate to produce a BigDecimal containing 1.05—this is equivalent to 1.0 + rate in line 19 of Fig. 5.6. The BigDecimal constant ONE represents the value 1. Class BigDecimal also provides the commonly used constants ZERO (0) and TEN (10).

Next, BigDecimal method pow is called on the preceding result to raise 1.05 to the power year—this is equivalent to passing 1.0 + rate and year to method Math.pow in line 19 of Fig. 5.6.

Finally, we call BigDecimal method multiply on the principal object, passing the preceding result as the argument. This returns a BigDecimal representing the amount on deposit at the end of the specified year.

-Rounding BigDecimal values(java.math.MathContext):BigDecimal also gives you control over how values are rounded—by default all calculations are exact and no rounding occurs. If you do not specify how to round BigDecimal values and a given value cannot be represented exactly—such as the result of 1 divided by 3, which is 0.3333333…—an ArithmeticException occurs. You can specify the rounding mode for BigDecimal by supplying a MathContext object to class BigDecimal’s constructor when you create a BigDecimal. You may also provide a MathContext to various BigDecimal methods that perform calculations. By default, each pre-configured MathContext uses so called “bankers rounding”.

Rounding and scaling BigDecimal values (Dynamic precision part 1): [dynamic\_precision\_part2](#dynamic_precision_part2) A BigDecimal’s scale is the number of digits to the right of its decimal point. If you need a BigDecimal rounded to a specific digit, you can call BigDecimal method setScale.

double floatingPointNumber = (double) numerator / denominator;

BigDecimal bd = new BigDecimal(floatingPointNumber).setScale(precision,

RoundingMode.HALF\_EVEN);

**-NumberFormat** (java.text.NumberFormat): US locale: “1,234.56” European locale: “1.234,56”

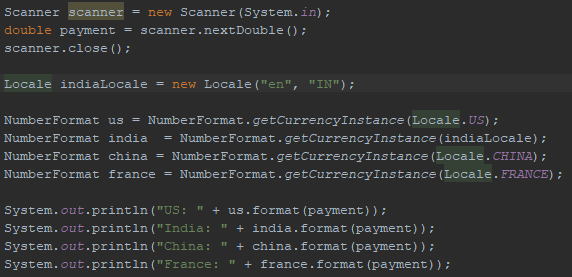
During each iteration of the loop, line 26

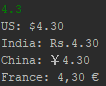
NumberFormat.getCurrencyInstance().format(amount)

evaluates as follows:

First, the expression uses NumberFormat’s static method getCurrencyInstance to get a NumberFormat that’s pre-configured to format numeric values as locale-specific currency Strings—for example, in the U.S. locale, the numeric value 1628.89 is formatted as $1,628.89 . Locale-specific formatting is an important part of internationalization—the process of customizing your applications for users’ various locales and spoken languages.

Next, the expression invokes NumberFormat method format (on the object returned by getCurrencyInstance) to perform the formatting of the amount value. Method format then returns the locale-specific String representation, rounded to two-digits to the right of the decimal point.





Or you could store us formatted version of an amount in a string and print that string.

String usAmount = us.format(payment);

System.out.println("US: " + usAmount);

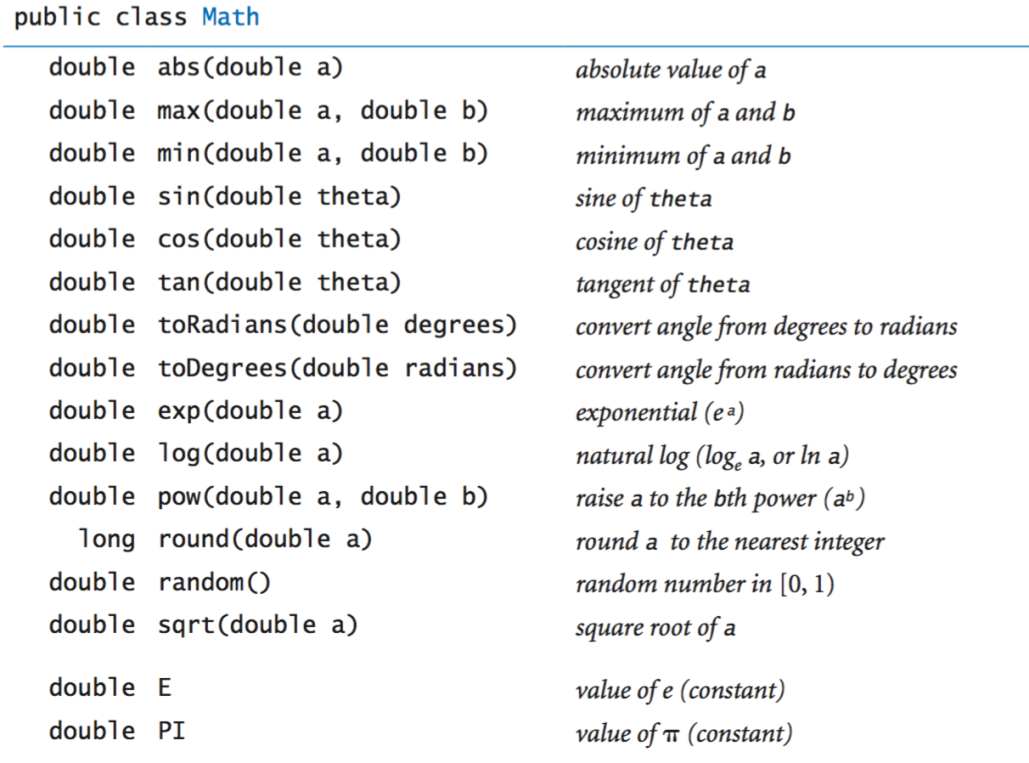
-Dynamic precision part 2: [dynamic\_precision\_part1](#dynamic_precision_part1) [dynamic precision part3](../Java%20Notes.docx#dynamic_precision_part3)

DecimalFormat df = new DecimalFormat();

df.setMaximumFractionDigits(2);

System.out.println(df.format(decimalNumber));

**-Math** <http://docs.oracle.com/javase/8/docs/technotes/guides/math/index.html>



**-Usage of Math:** Bir class’ın metodlarına nasıl class.metod şeklinde ulaşıyorsak, bir class’ın değişkenlerinede class.değişken diye ulaşırız.

Math.pow(x, 2);

valueOfE = Math.E;

**-System.exit:** Same as exit from C but this is portable. It is not a shell command.

<https://stackoverflow.com/questions/2434592/difference-in-system-exit0-system-exit-1-system-exit1-in-java>

System.exit(0);

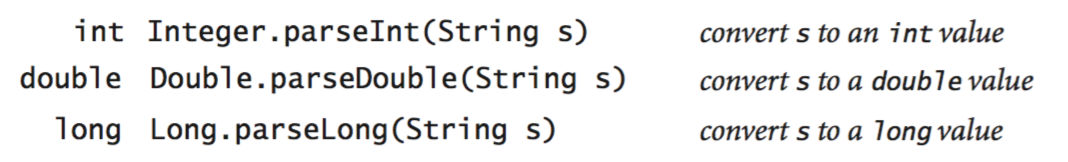
System.exit(-1);

System.exit(1);

- 0 when execution went fine;

- 1, -1, whatever != 0 when some error occurred, you can use different values for different kind of errors.

**-Integer:**



**-parseInt:**

String firstnumber = 123

int number1 = Integer.parseInt( firstNumber );

**-Calendar, LocalDate**(import java.util):

import java.util.Calendar;

...

return Calendar.getInstance().get(Calendar.YEAR) - birthYear;

Scanner input = new Scanner(System.in);

String year = input.nextLine();

String month = input.nextLine();

String day = input.nextLine();

// Java 7 Calendar class

Calendar c = Calendar.getInstance();

c.set(Integer.parseInt(year), Integer.parseInt(month), Integer.parseInt(day));

System.out.println(c.getDisplayName(Calendar.DAY\_OF\_WEEK, Calendar.LONG, Locale.US).toUpperCase());

// Java 8 LocalDate Class

LocalDate localDate = LocalDate.of(Integer.parseInt(year),Integer.parseInt(month),Integer.parseInt(day));

System.out.println(String.valueOf(localDate.getDayOfWeek()));  
  
**-Date:**

Date now = new Date();   
SimpleDateFormat formatter = new SimpleDateFormat( "MM/dd/yyyy" );   
System.out.println ( "The current date is: " + formatter.format( now ) );   
yields

"The current date is: 04/09/2009"

**-Time**(java.time)**:** Time API is the new, java 8 Date/Time API.