Advanced Java

LESSON 5: ENUMS AND TYPE SAFETY, NUMERIC WRAPPER CLASSES & DATES

Creative Commons

Attribution 4.0 International (CC BY 4.0)

This workforce product was funded by a grant awarded by the U.S. Department of Labor's Employment and Training Administration. The product was created by the grantee and does not necessarily reflect the official position of the U.S. Department of Labor. The U.S. Department of Labor makes no guarantees, warranties, or assurances of any kind, express or implied, with respect to such information, including any information on linked sites and including, but not limited to, accuracy of the information or its completeness, timeliness, usefulness, adequacy, continued availability, or ownership. This is an equal opportunity program. Assistive technologies are available upon request and include Voice/TTY (771 or 800-947-6644).

Agenda

Enums and Type Safety

Numeric Wrapper Classes

BigDecimal and BigInteger

Dates

Enums and Type Safety

Enums and Type Safety

Enumerated type (Enumeration)

- Are identifiers that behave as constants in the language.
- Have values that are different from each other, and that can be compared and assigned.
- Do not have any particular representation in the computer's memory. (Like the Boolean Data Types)
- Are used for known values:
  - Days of Week (MONDAY, TUESDAY, WEDNESDAY, THURSDAY)
  - Gender (MALE, FEMALE)
  - Payment Type (CASH, CREDIT, CHECK)

Enums and Type Safety

Enumerated type (Enumeration)

- Alternative – Using Integer Constants

```
class CurrencyDenom {
    public static final int PENNY = 1;
    public static final int NICKLE = 5;
    public static final int DIME = 10;
    public static final int QUARTER = 25;
}
```

- Not Type Safe. These variables are passed to methods as an int which means any value of type int may be passed to the same method even though they may not be legitimate values. (ex: 99)

- No Meaningful Print.
- (These would print the integer values not the coin name)
  - Payment Type (CASH, CREDIT, CHECK)

Enums and Type Safety

Alternative – String Values

```
public static void main(String[] args) {
       doIt("Nickle");
   }
   public static void doIt(String s) {
       int amt;
       switch(s) {
           case "Penny": amt=1; break;
           case "Nickle": amt=5; break;
           case "Dime": amt=10; break;
           case "Quarter": amt=25; break;
```

- Not Type Safe. These variables are passed to methods as a String which means any value of type String may be passed to the same method even though they may not be legitimate values. (ex: “Loonie”)

Enums and Type Safety

Java enum

- Enum is type like class and interface that is used to define a set of Enum constants.
- Enum constants are implicitly static and final which means you can not change there value once created.
- Enum in Java provides type-safety and can be used inside a switch statement like variables of type int.

Enums and Type Safety

Java enum

```
public enum Currency {

    PENNY, NICKLE, DIME, QUARTER

}
```

```
QUARTER
A Quarter
```

```
System.out.println(Currency.QUARTER);
Currency coins = Currency.QUARTER;
switch (coins) {
    case PENNY:
        System.out.println("A Penny");
        break;
    case NICKLE:
        System.out.println("A Nickle");
        break;
    case DIME:
        System.out.println("A Dime");
        break;
    case QUARTER:
        System.out.println("A Quarter");
        break;
}
```

Enums and Type Safety

Java enum

```
public enum Currency {
    PENNY(1), NICKLE(5), DIME(10),
                  QUARTER(25);
    private int value;

    private Currency(int value) {
        this.value = value;
    }
    public int getValue() {
        return value;
    }
}
```

```
Currency coin = Currency.NICKLE;


System.out.println(coin.ordinal());


System.out.println(coin.getValue());
```

In-Class Activity

1. Create a class that includes a main method that asks the user their favorite NFC team. Based upon the first three letters, the method should be able to identify the team based upon mascot or city.
2. There should be a second method with an argument of the enum type Team and should printout the name of the team with the name. City Mascot (Ex: Green Bay Packers, Chicago Bears)
3. Create the enum type Team.

Numeric Wrapper Classes

Numeric Wrapper Classes

Java enum

- There are situations when objects should be used in place of primitives, and the Java platform provides wrapper classes for each of the primitive data types.
- Classes:
  - Byte
  - Integer
  - Double
  - Short
  - Float
  - Long

Numeric Wrapper Classes

- Byte

  ```
  Byte.parseByte()
  ```

- Integer

  ```
  Integer.parseInt()
  ```

- Short

  ```
  Short.parseShort()
  ```

- Long

  ```
  Long.parseLong()
  ```

- Double

  ```
  Double.parseDouble()
  ```

- Float

  ```
  Float.parseFloat()
  ```

Numeric Wrapper Classes

Properties

- MAX_VALUE

Maximum number a variable of the data type may hold

- MIN_VALUE

Minimum number a variable of the data type may hold

- SIZE

Number of bits

- TYPE

Primitive datatype

Numeric Wrapper Classes

Properties

```
byte bVal = 0;
int val = Integer.parseInt("2345");
if ((val <= Byte.MAX_VALUE) && (val >= Byte.MIN_VALUE)) {
    bVal = (byte)val;
}
else {
    bVal = 0;
}
System.out.println(bVal);
```

Numeric Wrapper Classes

Autoboxing and Unboxing

- The automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes.

```
Integer val = 5;
Double dVal = 5.5;
double sum = val + dVal;

System.out.println(sum);
System.out.println(val);
System.out.println(dVal);
```

```
10.5
5
5.5
```

In-Class Activity

1. Create a class that includes a main method that asks the user for a number. The code should then determine which data type that uses the least amount of memory the number fits into.
2. The name of the data type should be printed out.

BigDecimal and BigInteger

BigDecimal and BigInteger

BigDecimal

- Has the ability to specify a scale, which represents the number of digits after the decimal place
- Has the ability to specify a rounding method.
- To set the number of digits after the decimal, use the `.setScale(scale)` method.
- It is good practice to also specify the rounding mode along with the scale by using `.setScale(scale, roundingMode)`.
- Package:

```
import java.math.BigDecimal;
```

BigDecimal and BigInteger

BigDecimal - Rounding

- ROUND_CEILING: Ceiling function
- ROUND_DOWN: Round towards zero
- ROUND_FLOOR: Floor function
- ROUND_HALF_UP: Round up if decimal >= .5
- ROUND_HALF_DOWN: Round up if decimal > .5
- ROUND_HALF_EVEN
- ROUND_UNNECESSARY

BigDecimal and BigInteger

BigDecimal - Immutability and Arithmetic

- BigDecimal numbers are immutable. What that means is that if you create a new BD with value "2.00", that object will remain "2.00" and can never be changed.
- So how do we do math then? The methods `.add()`, `.multiply()`, and so on all return a new BD value containing the result. For example, when you want to keep a running total of the order amount,

```
amount = amount.add( thisAmount );
```

- Do not do this;

```
amount.add( thisAmount );
```

BigDecimal and BigInteger

BigDecimal - Comparison

- It is important to never use the `.equals()` method to compare BigDecimals. That is because this equals function will compare the scale. If the scale is different , `.equals()` will return false, even if they are the same number mathematically.

```
BigDecimal a = new BigDecimal("2.00");
BigDecimal b = new BigDecimal("2.0");
print(a.equals(b)); // false
```

- Instead, we should use the `.compareTo()` and `.signum()` methods.

```
a.compareTo(b);
// returns (-1 if a < b), (0 if a == b), (1 if a > b)
a.signum();
// returns (-1 if a < 0), (0 if a == 0), (1 if a > 0)
```

BigDecimal and BigInteger

BigDecimal - Methods

- The BigDecimal class provides operations for arithmetic, scale manipulation, rounding, comparison, hashing, and format conversion. The toString() method provides a canonical representation of a BigDecimal.
- If no rounding mode is specified and the exact result cannot be represented, an exception is thrown; otherwise, calculations can be carried out to a chosen precision and rounding mode by supplying an appropriate MathContext object to the operation. In either case, eight rounding modes are provided for the control of rounding.

BigDecimal and BigInteger

BigDecimal - Methods - Constructors

- **`BigDecimal(BigInteger val)`** - Translates a BigInteger into a BigDecimal.
- **`BigDecimal(char[] in)`** - Translates a character array representation of a BigDecimal into a BigDecimal, accepting the same sequence of characters as the BigDecimal(String) constructor.
- **`BigDecimal(double val)`** - Translates a double into a BigDecimal which is the exact decimal representation of the double's binary floating-point value.
- **`BigDecimal(int val)`** - Translates an int into a BigDecimal.
- **`BigDecimal(long val)`** - Translates a long into a BigDecimal.
- **`BigDecimal(String val)`** - Translates the string representation of a BigDecimal into a BigDecimal.

BigDecimal and BigInteger

BigDecimal - Methods

- **`abs()`** – Returns a BigDecimal whose value is the absolute value of this BigDecimal.
- **`add(BigDecimal augend)`** – Returns a BigDecimal whose value is (this + augend).
- **`compareTo(BigDecimal val)`** – Compares this BigDecimal with the specified BigDecimal.
- **`divide(BigDecimal divisor)`** - Returns a BigDecimal whose value is (this / divisor) .
- **`divideAndRemainder(BigDecimal divisor) –`** Returns a two-element BigDecimal array containing the result of divideToIntegralValue followed by the result of remainder on the two operands.

BigDecimal and BigInteger

BigDecimal - Methods

- **`doubleValue()`** - Converts this BigDecimal to a double.
- **`equals(Object x)`** - Compares this BigDecimal with the specified Object for equality.
- **`floatValue()`** - Converts this BigDecimal to a float.
- **`intValue()`** - Converts this BigDecimal to an int.
- **`intValueExact()`** - Converts this BigDecimal to an int, checking for lost information.
- **`longValue()`** - Converts this BigDecimal to a long.
- **`longValueExact()`** - Converts this BigDecimal to a long, checking for lost information.

BigDecimal and BigInteger

BigDecimal - Methods

- `doubleValue()` - Converts this BigDecimal to a double.
- `max(BigDecimal val)` - Returns the maximum of this BigDecimal and val.
- `min(BigDecimal val)` - Returns the minimum of this BigDecimal and val.
- `movePointLeft(int n)` - Returns a BigDecimal which is equivalent to this one with the decimal point moved n places to the left.
- `movePointRight(int n)` - Returns a BigDecimal which is equivalent to this one with the decimal point moved n places to the right.

BigDecimal and BigInteger

BigDecimal - Methods

- `multiply(BigDecimal multiplicand)` - Returns a BigDecimal whose value is (this × multiplicand).
- `negate()` - Returns a BigDecimal whose value is (-this).
- `plus()` - Returns a BigDecimal whose value is (+this), and whose scale is this.scale().
- `pow(int n)` - Returns a BigDecimal whose value is (thisn), The power is computed exactly, to unlimited precision.
- `precision()` - Returns the precision of this BigDecimal.
- `remainder(BigDecimal divisor)` - Returns a BigDecimal whose value is (this % divisor).

BigDecimal and BigInteger

BigDecimal - Methods

- **`stripTrailingZeros()`** - Returns a BigDecimal which is numerically equal to this one but with any trailing zeros removed from the representation.
- **`subtract(BigDecimal subtrahend)`** - Returns a BigDecimal whose value is (this – subtrahend).
- **`toBigInteger()`** - Converts this BigDecimal to a BigInteger.
- **`toBigIntegerExact()`** - Converts this BigDecimal to a BigInteger, checking for lost information.
- **`toEngineeringString()`** - Returns a string representation of this BigDecimal, using engineering notation if an exponent is needed.

BigDecimal and BigInteger

BigDecimal - Methods

- **`toPlainString()`** - Returns a string representation of this BigDecimal without an exponent field.
- **`toString()`** - Returns the string representation of this BigDecimal, using scientific notation if an exponent is needed.
- **`unscaledValue()`** - Returns a BigInteger whose value is the unscaled value of this BigDecimal.
- **`valueOf(double val)`** - Translates a double into a BigDecimal, using the double's canonical string representation provided by the Double.toString(double) method.
- **`valueOf(long val)`** - Translates a long value into a BigDecimal with a scale of zero.

BigDecimal and BigInteger

BigDecimal

```
BigDecimal aDecimal = new BigDecimal(0.1950);
BigDecimal another = aDecimal.setScale(2,
                     aDecimal.ROUND_HALF_DOWN);
System.out.println("aDecimal: " + aDecimal);
System.out.println("another: " + another);
another = aDecimal.setScale(3,
                     aDecimal.ROUND_HALF_DOWN);
System.out.println("another Rounded: " + another);
another = another.add(BigDecimal.valueOf(10))
                 .divide( BigDecimal.valueOf(2));
System.out.println("another Divided: " + another);
```

```
aDecimal: 0.195000000000000006661338147750939242541790008544921875
another: 0.20
another Rounded: 0.195
another Divided: 5.0975
```

BigDecimal and BigInteger

BigInteger

- To work with integers that are larger than 64 bits (the size of a long), use java.math.BigInteger. This class represents unbounded integers and provides a number of methods for doing arithmetic with them.
- The problem with arithmetic using ints (or longs) is that, if the value becomes too large, Java saves only the low order 32 (64 for longs) bits and throws the rest away.
- Package:

```
import java.math.BigInteger;
```

BigDecimal and BigInteger

BigInteger - Constructors, and constants

- **`BigInteger(s)`** - Create BigInteger with decimal value represented by decimal String s.
- **`BigInteger.ONE`** - Predefined value 1.
- **`BigInteger.ZERO`** - Predefined value 0.
- **`BigInteger.valueOf(lng)`** - Use this factory method to create BigIntegers from numeric expressions. An int parameter will be automatically promoted to long.

BigDecimal and BigInteger

BigInteger - Arithmetic operations

- **`abs()`** - Returns BigInteger absolute value.
- **`b12.add(bi3)`** - Returns sum of bi2 and bi3.
- **`bi2.divide(bi3)`** - Returns division of bi2 and bi3.
- **`bi2.divideAndRemainder(bi3)`** - Returns array of two BigIntegers representing the result of division and remainder of bi2 and bi3.
- **`bi2.gcd(bi3)`** - Returns greatest common divisor of bi2 and bi3.
- **`bi2.max(bi3)`** - Returns maximum of bi2 and bi3.
- **`bi2.min(bi3)`** - Returns minimum of bi2 and bi3.

BigDecimal and BigInteger

BigInteger - Arithmetic operations

- **`bi2.mod(bi3)`** - Returns remainder after dividing bi2 by bi3.
- **`bi2.multiply(bi3)`** - Returns product of bi2 and bi3.
- **`bi2.pow(bi3) - Returns bi2 to the bi3 power.`**
- **`bi2.remainder(bi3)`** - Returns remainder of dividing bi2 by bi3. May be negative .
- **`signum()`** - -1 for neg numbers, 0 for zero, and +1 for positive.
- **`bi2.subtract(bi3)`** - Returns bi2 - bi3.

BigDecimal and BigInteger

BigInteger - Conversion to other types

- **`doubleValue()`** - Returns double value.
- **`floatValue()`** - Returns float value equivalent.
- **`intValue()`** - Returns int value equivalent.
- **`longValue()`** - Returns long value equivalent.
- **`toString()`** - Returns decimal string representation.
- **`bi1.compareTo(bi2)`** - Returns negative number if bi1<bi2, 0 if bi1==bi2, or positive number if bi1>bi2.

BigDecimal and BigInteger

```
BigInteger bigInt1 = new BigInteger ("123456789");
BigInteger bigInt2 = new BigInteger ("9876543");
System.out.println("Result is => " + bigInt1.multiply(bigInt2));
BigInteger good = BigInteger.valueOf(2000000000);
System.out.println("good.add(BigInteger.ONE) = " +
       good.add(BigInteger.ONE));
System.out.println("good.multiply(BigInteger.valueOf(3)) = " +
       good.multiply(BigInteger.valueOf(3)));
System.out.println("good.multiply(BigInteger.valueOf(4)) = " +
       good.multiply(BigInteger.valueOf(4)));
```

```
Result is => 1219326285200427
good.add(BigInteger.ONE) = 2000000001
good.multiply(BigInteger.valueOf(3)) = 6000000000
good.multiply(BigInteger.valueOf(4)) = 8000000000
```

In-Class Activity

1. Create a class that includes a main method that calculates how many letters everyone in Wisconsin would have to write if they wrote everyone in California a letter and how many copies of those letters would have to be made for everyone in Texas to have a copy. Print the number of copies that would have to be made for Texas and how much it would cost at $3.23 each to copy them.
2. Populations:

Wisconsin Total	5,726,398
California	38,041,430
Texas	26,059,203

Dates

Date

Date and Calendar Classes

- Share the same fundamental concept (both represent an instant in time and are wrappers around an underlying long value).
- Use Calendar as a calculator which, when given Date and TimeZone objects, will perform calculations.
- Use SimpleDateFormat together with TimeZone and Date to generate display Strings.

Date

Date Classes

- Share the same fundamental concept (both represent an instant in time and are wrappers around an underlying long value).
- Use Calendar as a calculator which, when given Date and TimeZone objects, will perform calculations.
- Use SimpleDateFormat together with TimeZone and Date to generate display Strings.

Date

Date Classes

- Date objects represent dates and times. You cannot display or print a Date object without first converting it to a String that is in the proper format.

```
Date today;
String dateOut;
DateFormat dateFormatter;
dateFormatter =
      DateFormat.getDateInstance(DateFormat.DEFAULT, Locale.US);
today = new Date();
dateOut = dateFormatter.format(today);
System.out.println(dateOut);
```

```
Feb 4, 2013
```

Date

GregorianCalendar

- GregorianCalendar is a hybrid calendar that supports both the Julian and Gregorian calendar systems with the support of a single discontinuity.

Date

GregorianCalendar – Methods - Constructors

- `GregorianCalendar()` - Constructs a default GregorianCalendar using the current time in the default time zone with the default locale.
- `GregorianCalendar(int year, int month, int dayOfMonth)`
- `GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute)`
- `GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute, int second)`
- `GregorianCalendar(Locale aLocale)`
- `GregorianCalendar(TimeZone zone)`

Date

GregorianCalendar – Methods - Constructors

- **`GregorianCalendar()`** - Constructs a default GregorianCalendar using the current time in the default time zone with the default locale.
- **`getTimeZone`**() - Gets the time zone.
- **`hashCode()`** - Generates the hash code for this GregorianCalendar object.
- **`isLeapYear(int year)`** - Determines if the given year is a leap year.
- **`roll(int field, boolean up)`** - Adds or subtracts (up/down) a single unit of time on the given time field without changing larger fields.

Date

GregorianCalendar

```
/** The date at the end of the last century */
Date bDay = new GregorianCalendar(1970,5,8).getTime();
/** Today's date */
Date today = new Date();
// Get msec from each, and subtract.
long diff = today.getTime() - bDay.getTime();
System.out.println("As of " + today + " Matt is " +
     (diff / (1000 * 60 * 60 * 24)) + " days old.");
```

```
As of Mon Feb 04 20:42:16 CST 2013 Matt is 15582 days old.
As of Mon Feb 04 20:42:16 CST 2013 Matt is 15582 days old.
```

Date

Simple Date Formatter

- To display a date and time in the same String, create the formatter with the getDateTimeInstance method. The first parameter is the date style, and the second is the time style. The third parameter is the Locale .

Date

Simple Date Formatter - Date

```
Date today;
String dateOut;
DateFormat dateFormatter;
dateFormatter = DateFormat.getDateInstance(DateFormat.DEFAULT,
        Locale.US);
today = new Date();
dateOut = dateFormatter.format(today);
System.out.println(dateOut);
```

```
Feb 4, 2013
```

Date

Simple Date Formatter - Time

```
Date today;
String dateOut;
DateFormat timeFormatter =
    DateFormat.getTimeInstance(DateFormat.DEFAULT, Locale.US);
today = new Date();
dateOut = timeFormatter.format(today);
System.out.println(dateOut);
```

```
8:57:18 PM
```

Date

Simple Date Formatter – Date & Time

```
Date today;
String dateOut;
DateFormat formatter = DateFormat.getDateTimeInstance(
                           DateFormat.LONG,
                           DateFormat.LONG,
                           Locale.US);
today = new Date();
dateOut = formatter.format(today);
System.out.println(dateOut);
```

```
February 4, 2013 8:59:19 PM CST
```

In-Class Activity

1. Create a class that includes a main method that calculates how many days, months and years since U2 released the Joshua Tree album.
2. Rattle and Hum was produced the next year. Print out when would be released if The Joshua Tree was released today and it took U2 the same amount of time to release the next album.
3. U2:

The Joshua Tree	9 March 1987
Rattle and Hum	10 October 1988