Enum, Number, String

Enum

...

Enum

An ***enum*** *type* is a special data type that enables for a variable to be a set of predefined constants.

```
public enum Gender {
    MALE,
    FEMALE;
}

public enum Season {
    WINTER,
    SPRING,
    SUMMER,
    FALL
}
```

The enum class body can include methods and other fields.

```
public enum Planet {
    MERCURY(3.303e+23, 2.4397e6),
    VENUS  (4.869e+24, 6.0518e6),
    EARTH  (5.976e+24, 6.37814e6),
    ...
    private final double mass;
    private final double radius;
    private Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
    public double getMass() {
        return mass;
    }
    public double getSurfaceGravity() {
        return G * mass / (radius * radius);
    }
}
```

Enum example

```
public enum Direction {
    NORTH(0, 1),
    EAST(1, 0),
    SOUTH(0, -1),
    WEST(-1, 0);
    private final int x;
    private final int y;
    private Direction(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
}
...
public void move(Direction direction) {
    currentX += direction.getX();
    currentY += direction.getY();
}
```

Enum

- *All* enums implicitly extend java.lang.Enum.
- All enum constants implicitly have `public static final modifier`
- You cannot create instance of enum with **new** operator
- You cannot extend enum

Enum

- Non static methods of enum:
  - `ordinal()` - Returns the ordinal of this enumeration constant (its position in its enum declaration, where the initial constant is assigned an ordinal of zero).
  - `compareTo()` - compares this enum with the specified object for order

- Static enum methods:
  - `values()` – All the constants of an enum type can be obtained by calling this method
  - `valueOf(String name)` – Returns the enum constant of the specified enum type with the specified name

Annotation

- Annotation is a form of metadata, provide data about a program that is not part of the program itself

- Annotations have a number of uses, among them:
  - Information for the compiler
  - Compile-time and deployment-time processing
  - Runtime processing

Predefined Annotation Types

- @Deprecated

```
/**
 * @deprecated explanation of why it was deprecated
 */
@Deprecated
static void deprecatedMethod() {
}
```

- @Override

```
@Override
int overriddenMethod() {
}
```

- @SuppressWarnings

```
@SuppressWarnings("deprecation")
void useDeprecatedMethod() {
    deprecatedMethod();
}
```

Number

...

Math

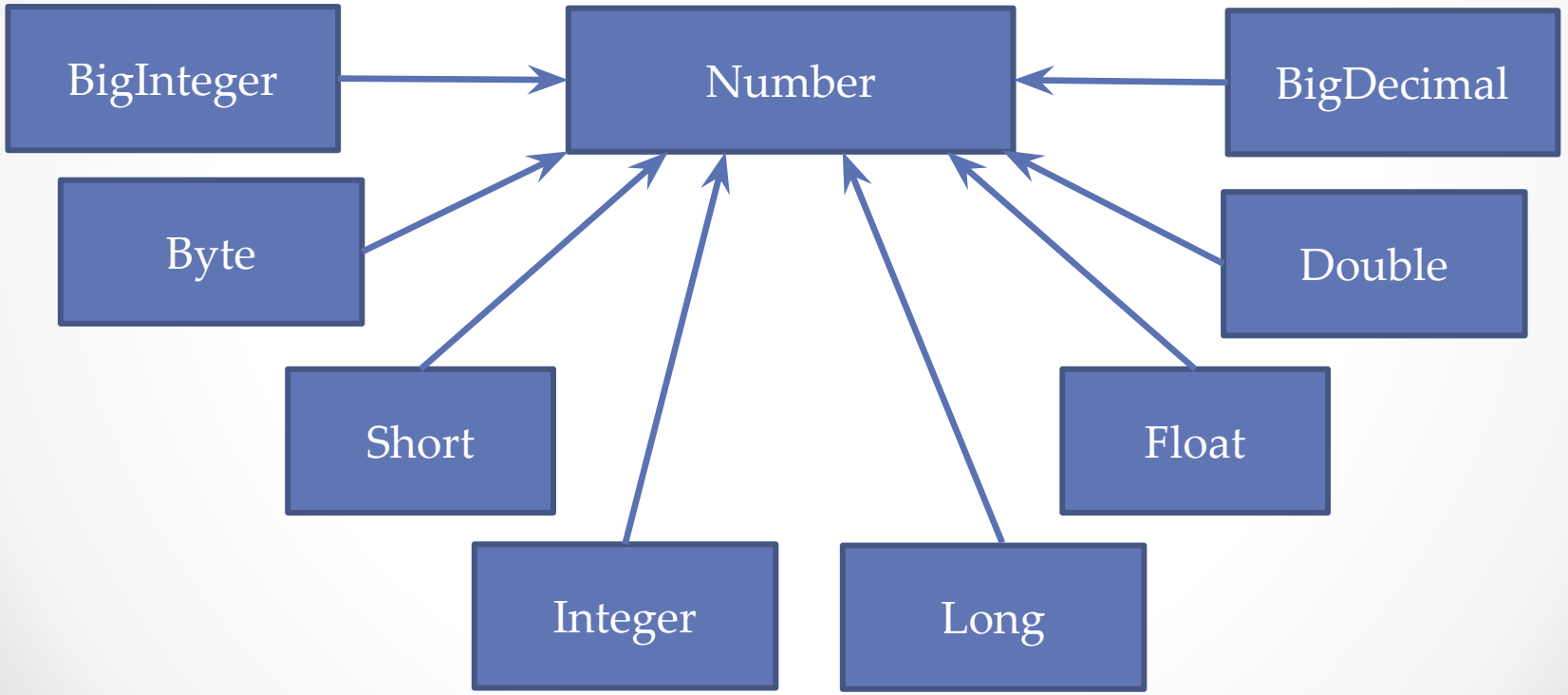
Constants	
Math.E	Math.PI
Static methods	
double abs(double d)	double min(double arg1, double arg2)
float abs(float f)	float min(float arg1, float arg2)
int abs(int i)	int min(int arg1, int arg2)
long abs(long lng)	long min(long arg1, long arg2)
double ceil(double d)	double max(double arg1, double arg2)
double floor(double d)	float max(float arg1, float arg2)
double rint(double d)	int max(int arg1, int arg2)
long round(double d)	long max(long arg1, long arg2)
int round(float f)	

Math

double exp(double d)	double sin(double d)
double log(double d)	double cos(double d)
double pow(double base, double exponent)	double tan(double d)
double sqrt(double d)	double asin(double d)
	double acos(double d)
	double atan(double d)
	double atan2(double y, double x)
	double toDegrees(double d)
	double toRadians(double d)

Number classes

All of the numeric wrapper classes are subclasses of the abstract class



Numbers

There are three reasons that you might use a Number object rather than a primitive:

- As an argument of a method that expects an object (often used when manipulating collections of numbers).
- To use constants defined by the class, such as MIN_VALUE and MAX_VALUE, that provide the upper and lower bounds of the data type.
- To use class methods for converting values to and from other primitive types, for converting to and from strings, and for converting between number systems (decimal, octal, hexadecimal, binary).

Number methods

Convertsion	Comparison
byte byteValue()	int compareTo(Byte anotherByte)
short shortValue()	int compareTo(Short anotherShort)
int intValue()	int compareTo(Integer anotherInteger)
long longValue()	int compareTo(Long anotherLong)
float floatValue()	int compareTo(Float anotherFloat)
double doubleValue()	int compareTo(Double anotherDouble)

Integer

```
public static Integer decode(String nm)

public static int parseInt(String s)

public static int parseInt(String s, int radix)

public static String toString(int i)

public static Integer valueOf(int i)

public static Integer valueOf(String s)

public static Integer valueOf(String s, int radix)
```

Double

static int	MAX_EXPONENT Maximum exponent a finite double variable may have.
static double	MAX_VALUE A constant holding the largest positive finite value of type double, $(2-2^{-52})\cdot 2^{1023}$.
static int	MIN_EXPONENT Minimum exponent a normalized double variable may have.
static double	MIN_NORMAL A constant holding the smallest positive normal value of type double, 2^{-1022}.
static double	MIN_VALUE A constant holding the smallest positive nonzero value of type double, 2^{-1074}.
static double	NaN A constant holding a Not-a-Number (NaN) value of type double.
static double	NEGATIVE_INFINITY A constant holding the negative infinity of type double.
static double	POSITIVE_INFINITY A constant holding the positive infinity of type double.
static int	SIZE The number of bits used to represent a double value.

BigInteger

- handling very large integers
- provides analogues to all of Java's primitive integer operators
- provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation

```
public BigInteger(String val)
public BigInteger(byte[] val)
public static BigInteger valueOf(long val)
```

BigInteger methods

```
public BigInteger add(BigInteger val)
public BigInteger subtract(BigInteger val)
public BigInteger multiply(BigInteger val)
public BigInteger divide(BigInteger val)
public BigInteger mod(BigInteger m)
public BigInteger pow(int exponent)

public BigInteger abs()
public BigInteger negate()
public int signum()

public BigInteger and(BigInteger val)
public BigInteger or(BigInteger val)
public BigInteger xor(BigInteger val)
public BigInteger not()
```

BigDecimal

- java.math.BigDecimal class provides operations for arithmetic, scale manipulation, rounding, comparison, hashing, and format conversion.

- BigDecimal is immutable

```
public BigDecimal(String val)
public BigDecimal(BigInteger val)
public BigDecimal(double val)
public static BigDecimal valueOf(double val)
public static BigDecimal valueOf(long val)
```

BigDecimal methods

```
public BigDecimal add(BigDecimal augend)
public BigDecimal subtract(BigDecimal subtrahend)
public BigDecimal multiply(BigDecimal multiplicand,
                           MathContext mc)
public BigDecimal divide(BigDecimal divisor,
                         MathContext mc)
public BigDecimal pow(int n, MathContext mc)

public BigDecimal abs()
public int signum()

public int scale()
public int precision()
```

Character - wrapper for char

```
public static boolean isLetter(char ch)
public static boolean isDigit(char ch)
public static boolean isWhitespace(char ch)
public static boolean isUpperCase(char ch)
public static boolean isLowerCase(char ch)

public static char toUpperCase(char ch)
public static char toLowerCase(char ch)
public static String toString(char c)
```

Autoboxing

- ***Autoboxing*** is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes

- Converting an object of a wrapper type (Integer) to its corresponding primitive (int) value is called ***unboxing***.

Autoboxing

- The Java compiler applies autoboxing when a primitive value is:
  - Passed as a parameter to a method that expects an object of the corresponding wrapper class.
  - Assigned to a variable of the corresponding wrapper class.
- The Java compiler applies unboxing when an object of a wrapper class is
  - Passed as a parameter to a method that expects a value of the corresponding primitive type.
  - Assigned to a variable of the corresponding primitive type.

Wrapper classes

Primitive	Wrappers
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Autoboxing

```
Integer integer = 1; // Integer.valueOf(1)
int i = integer;     // integer.intValue()

Character character = 'a';
char c = character;

Double value = 1.0;
double d = value;

Byte byteValue = null;
byte b = byteValue; // NullPointerException
```

Autoboxing

```
public static List<Integer> asList(final int[] a) {
    return new AbstractList<Integer>() {
        public Integer get(int i) {
            return a[i];
        }

        public Integer set(int i, Integer value) {
            Integer old = a[i];
            a[i] = value;
            return old;
        }
        public int size() {
            return a.length;
        }
    };
}
```

String

...

String

```
public final class String
        implements java.io.Serializable,
                   Comparable<String>, CharSequence
```

- String creation

```
String greeting = "Hello world!";

char[] array = {'h', 'e', 'l', 'l', 'o'};
String helloString = new String(array);
```

Working with strings

```
String s = "Some text";
s.length();
"Text".length();
```

- String concatenation

```
s = s.concat("Additional text");
s = "Text".concat("Another text");
s = "Text" + "Another text";
```

Converting string to number

- Wrapper classes

```
Integer value = Integer.valueOf("1");

Double value = Double.valueOf("1.0");
```

- Primitive types

```
int value = Integer.parseInt("1");

double value = Double.parseDouble("1.0");
```

Converting number to string

- String

```
String.valueOf(1);

String.valueOf(1.0);
```

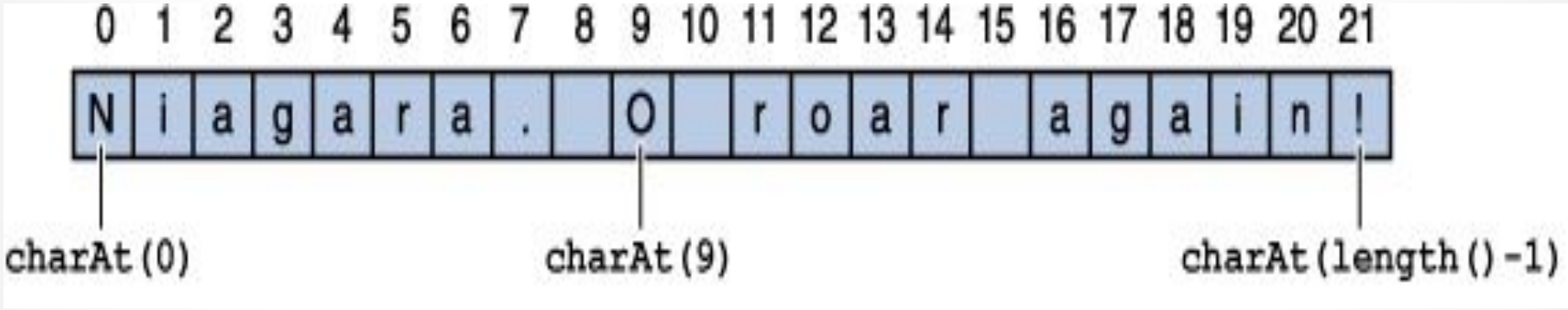
- Number classes

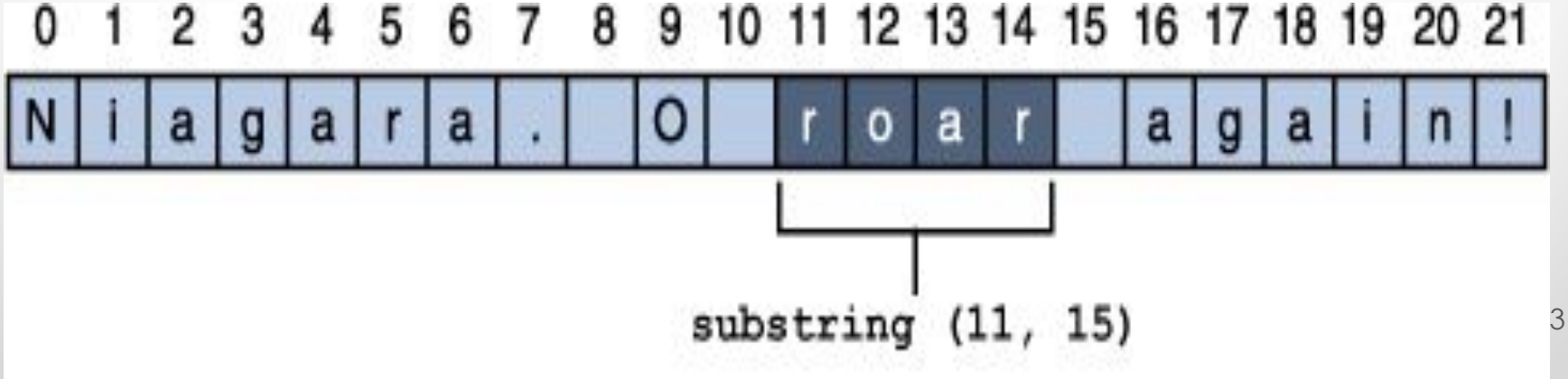
```
Integer.toString(1);

Double.toString(1.0);
```

Getting Characters and Substrings by Index

```
String text = "Niagara. O roar again!";
char aChar = text.charAt(9);
String roar = text.substring(11, 15);
```





String methods

```
public String[] split(String regex)

public String[] split(String regex, int limit)

public CharSequence subSequence(int beginIndex,
                                int endIndex)

public String trim()

public String toLowerCase()

public String toUpperCase()
```

Searching for Characters and Substrings in a String

```
public int indexOf(int ch)
public int indexOf(int ch, int fromIndex)
public int indexOf(String str)
public int indexOf(String str, int fromIndex)

public int lastIndexOf(int ch)
public int lastIndexOf(int ch, int fromIndex)
public int lastIndexOf(String str)
public int lastIndexOf(String str, int fromIndex)

public boolean contains(CharSequence s)
```

Replacing Characters and Substrings into a String

```
public String replace(char oldChar, char newChar)

public String replace(CharSequence target,
                      CharSequence replacement)

public String replaceAll(String regex,
                         String replacement)

public String replaceFirst(String regex,
                           String replacement)
```

Comparing Strings and Portions of Strings

```
public boolean endsWith(String suffix)
public boolean startsWith(String prefix)

public int compareTo(String anotherString)
public int compareToIgnoreCase(String str)

public boolean equals(Object anObject)
public boolean equalsIgnoreCase(String str)

public boolean matches(String regex)
```

String immutability

- String objects are immutable
  - String is not changed:

```
s.concat("big");
```

  - Currently "s" refers to new String object that was created during concantenation

```
s = s.concat("big");
```

String modification

- Each String modification creates new String.
- N Strings will be created

```
String s = "";
for (int i = 0; i < n; i++) {
    s += "*";
}
```

StringBuilder

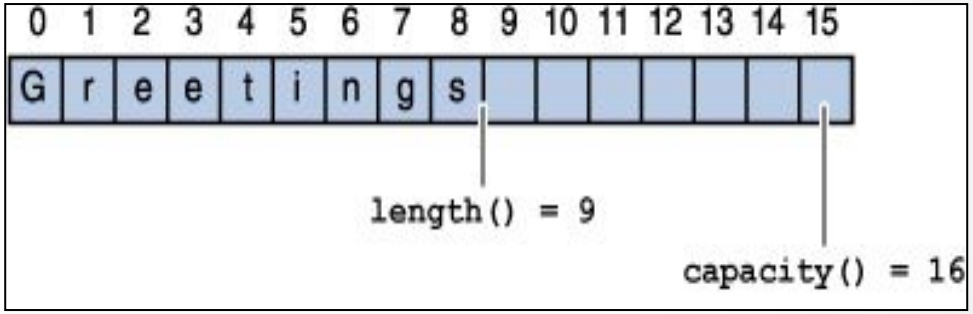
- Like String objects, except that they can be modified.
- Strings should always be used unless string builders offer an advantage in terms of simpler code or better performance.
- if you need to concatenate a large number of strings, appending to a StringBuilder is more efficient

```
StringBuilder builder = new StringBuilder();
for (int i = 0; i < n; i++) {
    builder.append("*");
}
```

StringBuilder

```
public StringBuilder()
public StringBuilder(int capacity)
public StringBuilder(String str)
public StringBuilder(CharSequence seq)
```

- Length and capacity




```
public int length()
public void setLength(int newLength)
public int capacity()
public void ensureCapacity(int minimumCapacity)
```

StringBuilder methods

```
public StringBuilder append(Object obj)

public StringBuilder delete(int start, int end)

public StringBuilder deleteCharAt(int index)

public StringBuilder insert(int offset, Object obj)

public StringBuilder replace(int start, int end, String str)

public void setCharAt(int index, char ch)

public StringBuilder reverse()

public String toString()
```

String & StringBuilder

number of words	Length	String		StringBuilder	
		Time	Memory	Time	Memory
3	15	0.163	30	0.22	16
4	20	0.252	50	0.373	48
5	25	0.336	75	0.39	48
6	30	0.464	105	0.463	48
7	35	0.565	140	0.591	112
15	75	1.779	600	1.125	240
20	100	2.697	1050	1.354	240
25	125	3.811	1625	1.571	240
50	250	11.45	6375	3.03	496
90	450	32.13	20475	5.419	1008

Date

...

Working with dates

- Main classes
  - java.util.Date
  - java.util.Calendar
- For Database
  - java.sql.Timestamp
  - java.sql.Date
  - java.sql.Time

java.util.Date

- Constructors

```
public Date()
public Date(long date)
```

- Main methods

```
public boolean before(Date when)
public boolean after(Date when)
public int compareTo(Date anotherDate)
public long getTime()
public void setTime(long time)
```

- The most part of other methods are deprecated

Calendar

- Main methods

```
public static Calendar getInstance()

public final void setTime(Date date)
public final void set(int year, int month, int date)
public void set(int field, int value)

public final Date getTime()
public int get(int field)

public void add(int field, int amount);
```

- Constants:
  - *ERA, YEAR, MONTH, WEEK_OF_YEAR, WEEK_OF_MONTH, DAY_OF_MONTH, DAY_OF_YEAR, DAY_OF_WEEK, AM_PM, HOUR, HOUR_OF_DAY, MINUTE, SECOND, MILLISECOND*

DateFormat

```
DateFormat dateFormat
        = new SimpleDateFormat("dd.MM.yyyy");



Date date = dateFormat.parse("04.06.2012");



String text = dateFormat.format(new Date());
```

DateFormat

Буква	Значение	Пример
G	Эра	AD
y	Год	1996; 96
M	Месяц	July; Jul; 07
w	Неделя в году	27
W	Неделя в месяце	2
D	День года	189
d	День месяца	10
F	День недели	2
E	День недели	Tuesday; Tue
a	До полудня/После полудня	PM
H	Час (0-23)	0
k	Час (1-24)	24
K	Час (0-11)	0
h	Час (1-12)	12
m	Минута	30
s	Секунда	55
S	Миллисекунда	978
z	Часовой пояс	Pacific Standard Time; PST; GMT-08:00
Z	Часовой пояс	-0800


Questions

Task №3 – Общие требования

Общие требования:

1. Код должен быть отформатирован и соответствовать Java Code Convention.
2. Решение поставленной задачи, должно быть реализовано в классе, который находится в пакете ***com.epam.firstname_lastname.java.lesson_number.task_number***, *например:* ***com.epam.anton_ostrenko.java.lesson3.task3***.
3. Класс, который содержит main-метод, должен иметь осмысленное название. Внутри метода main создайте объект его класса, у которого вызовете метод, являющийся стартовым для решения вашей задачи.
4. По возможности документируйте код.

Task №3 – Задание №1

В Учебном Центре компании проходят обучение студенты. Каждый студент проходит обучение по определенной индивидуальной программе. Программа обучения состоит из набора курсов, которые студент проходит последовательно. Каждый курс имеет определенную длительность.

Приложение должно позволять:

- определить относительно текущей даты закончил студент изучение программы или нет.
- рассчитать, сколько дней и часов осталось студенту до окончания программы или сколько дней и часов назад студент закончил изучение программы обучения.

Task №3 – список данных о студентах:

STUDENT: Ivanov Ivan
CURRICULUM: J2EE Developer
START_DATE: <указать дату получения задания>

COURSE	DURATION (hrs)
1. Технология Java Servlets	16
2. Struts Framework	24

STUDENT: Petrov Petr
CURRICULUM: Java Developer
START_DATE: <указать дату получения задания>

COURSE	DURATION (hrs)
1. Обзор технологий Java	8
2. Библиотека JFC/Swing	16
3. Технология JDBC	16

Непосредственно в коде следует прописать дату получения данного задания

Task №3 – список данных о студентах:

STUDENT: Ivanov Ivan
CURRICULUM: J2EE Developer
START_DATE: <указать дату получения задания>

COURSE	DURATION (hrs)
1. Технология Java Servlets	16
2. Struts Framework	24

STUDENT: Petrov Petr
CURRICULUM: Java Developer
START_DATE: <указать дату получения задания>

COURSE	DURATION (hrs)
1. Обзор технологий Java	8
2. Библиотека JFC/Swing	16
3. Технология JDBC	16

Непосредственно в коде следует прописать дату получения данного задания

Task №3 – Задание 1

Условия:

Учебными считаются все дни недели при условии 8-ми часового учебного дня с 10 до 18.

Ввод/Вывод:

Результат расчета вывести в консоль с указанием имени студента и изучаемой программы.

Пример вывода.

Ivanov Ivan (Java Developer) - Обучение не закончено. До окончания осталось 1 д 6 ч.

Petrov Petr (J2EE Developer) - Обучение закончено. После окончания прошло 3 ч.

Расчет этого времени учитывает длительность учебного дня.

Task №3 – Задание 1

Условия:

Учебными считаются все дни недели при условии 8-ми часового учебного дня с 10 до 18.

Ввод/Вывод:

Результат расчета вывести в консоль с указанием имени студента и изучаемой программы.

Пример вывода.

Ivanov Ivan (Java Developer) - Обучение не закончено. До окончания осталось 1 д 6 ч.

Petrov Petr (J2EE Developer) - Обучение закончено. После окончания прошло 3 ч.

Расчет этого времени учитывает длительность учебного дня.

Task №3 – Задание 1

2. Вывести подробный отчет по обучению: ФИО, рабочее время (с 10 до 18), название программы, длительность программы в часах, дата старта, дата завершения, сколько прошло/осталось до завершения.

Выбор варианта запуска осуществляется входящим параметром (нет параметра или параметр 0 – сокращенный вид отчета, иначе – подробный.

Task №3 – Дополнительное задание

Реализовать template generator. Например,

есть шаблон: “Hello, ${name}” , и значение name=”Reader” , TemplateGenerator должен вернуть “Hello Reader”

“Hello, ${name}”, и значение для name не было установлено -> Error

“${one}, ${two}, ${three}”, значения one=”1”, two=”${2}”, three = 3

“1, ${2}, 3”

Шаблон и значения вводяться с консоли.

Пример,

Введите шаблон:

“${greeting}, ${name}”

Введите переменные:

greeting=Hi, name=Petro

Результат: Hi, Petro

Task №3 – Критерии оценки

- Код приложения должен быть отформатирован в едином стиле и соответствовать Java Code Convention – 1 балл
- При выполнении задания должны быть использованы Numbers, Strings, Dates – 2 балла
- В задании должны быть корректно выполнены все пункты – 5 баллов
- Код легко читаемый, в коде отсутствует «кривизна», все методы и переменные поименованы понятными смысловыми именами – 2 балла