Primitive Type Wrappers

Each primitive type has its object wrapper. Although usage of primitive types is more efficient, there are some situations where application of their object wrappers is either more convenient or just inevitable.

Primitive type	Size [bits]	Wrapper class
`boolean`	-	`Boolean`
`char`	16	`Character`
`byte`	8	`Byte`
`short`	16	`Short`
`int`	32	`Integer`
`long`	64	`Long`
`float`	32	`Float`
`double`	64	`Double`
`void`	-	`Void`

Instances of wrapper classes represent immutable values.

Primitive Type Wrappers (cont.)

There is a common abstract superclass of `Byte`, `Short`, `Integer`, `Long`, `Float` and `Double` called `Number`.

Wrapper classes contain also many useful static methods for manipulation with values of the given primitive type:

- methods that transform the values of the primitive type to strings
- methods that transform strings to the given primitive type

Wrapper classes for numeric types also contain static constants `MIN_VALUE` and `MAX_VALUE` representing the minimal and maximal value of the given numeric type.

```
short x = Short.MIN_VALUE;  // x = -32768
```

Class Character

Class `Character` contains many static methods for testing if a character belongs to the given category of characters:

- static boolean isLowerCase(char ch)
- static boolean isUpperCase(char ch)
- static boolean isDigit(char ch)
- static boolean isDefined(char ch)
- static boolean isLetter(char ch)
- static boolean isLetterOrDigit(char ch)
- static boolean isSpaceChar(char ch)
- static boolean isWhitespace(char ch)
- static boolean isISOControl(char ch)
- . . .

Java uses 16-bit character encoding called Unicode. More information about Unicode can be found at `http://www.unicode.org`.

ASCII Table

The first 128 characters of Unicode are the same as in the ASCII table.

ASCII – American Standard Code of Information Interchange

0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Manipulation With Characters

Some examples of manipulation with characters. Notice how properties of ASCII are used.

- Transformation of a decimal digit to the numerical value:

```
static int decToNum(char c) {
    if (c >= '0' && c <= '9') return c - '0';
    return -1;
}
```

- Transformation of a hexadecimal digit to the numerical value:

```
static int hexToNum(char c) {
    if (c >= '0' && c <= '9')
        return c - '0';
    else if (c >= 'A' && c <= 'F')
        return c - 'A' + 10;
    else if (c >= 'a' && c <= 'f')
        return c - 'a' + 10;
    return -1;
}
```

Class Character (cont.)

Class Character contains static methods for transformation of characters to lower-case and upper-case:

- static char toLowerCase(char ch)
- static char toUpperCase(char ch)

These methods work for all Unicode characters.

Simplified version of `toLowerCase()` working only on ASCII characters could look like this:

```
static char toLowerCase(char c) {
    if (c >= 'A' && c <= 'Z') {
        return (char)(c - 'A' + 'a');
    }
    return c;
}
```

Class Integer

Class Integer contains the following static method for transformation of integer values to strings:

- static String toString(int i, int radix)
- static String toString(int i)
- static String toOctalString(int i)
- static String toHexString(int i)
- static String toBinaryString(int i)

Class Long contains similar methods that work with type **long**.

We can use the following methods to transform a string into an integer:

- static int parseInt(String s, int radix)
- static int parseInt(String s)
- static Integer valueOf(String s, int radix)
- static Integer valueOf(String s)
- static Integer decode(String nm)

Class Number

The abstract class `Number` defines the following (instance) methods:

- byte byteValue()
- short shortValue()
- int intValue()
- long longValue()
- float floatValue()
- double doubleValue()

Each subclass of `Number` (`Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`) implements these methods.

Subclasses of `Number` implemet static methods for transformation of strings into the corresponding primitive type:

- Class **Byte**:
  - static byte parseByte(String s, int radix)
  - static byte parseByte(String s)
- . . .

Classes Float and Double

Classes `Float` and `Double` contain static fields representing some special values of types **`float`** and **`double`**:

- NaN – Not-a-Number
- `NEGATIVE_INFINITY`
- `POSITIVE_INFINITY`

These values can be used as any other **`float`** or **`double`** values.

There are also static methods that allow to test these special values:

- static boolean isNaN(float v)
- static boolean isInfinite(float v)
- static boolean isNaN(double v)
- static boolean isInfinite(double v)

Class Math

The class **java.lang.Math** contains methods for performing basic numeric operations such as:

- exponential
- logarithm
- square root
- trigonometric functions

The class **Math** contains two important static constants:

- E – Euler number, the base of the natural logarithms (2.71828...)
- PI – number *pi* (3.14159...)

It is not possible to create instances of **Math**.

All methods are static.

Class Math (cont.)

Overview of methods:

- static double exp(double a)
- static double log(double a)
- static double sqrt(double a)
- static double pow(double a, double b)
- static double sin(double a)
- static double cos(double a)
- static double tan(double a)
- static double asin(double a)
- static double acos(double a)
- static double atan(double a)
- static double atan2(double y, double x)
- static double toRadians(double angdeg)
- static double toDegrees(double angrad)

Class Math (cont.)

Other methods:

- static double ceil(double a)
- static double floor(double a)
- static double rint(double a)
- static int round(float a)
- static long round(double a)
- static double random()
- static int abs(int a)
- static long abs(long a)
- static float abs(float a)
- static double abs(double a)
- static int max(int a, int b)
- static int min(int a, int b)
- . . .

Class Math (cont.)

Example of usage of mathematical functions:

```
double start = 0.0;
double end = Math.PI * 2.0;
double step = 0.05;
for (double x = start; x <= end; x += step) {
    double y1 = Math.sin(x);
    double y2 = Math.cos(x);
    System.out.print("x=" + x);
    System.out.print("  sin(x)=" + y1);
    System.out.println("  cos(x)=" + y2);
}
```

One possible way how numbers can be rounded (except using the method round()):

```
double a;
  ...
int i = (int)(a + 0.5);
```

Method main()

At least one class in a program must contain the static method

```
public static void main(String[] args)
```

Such classes can be used to start programs:

```
$ java MyClass
```

The method `main()` usually creates instances of other classes and calls their methods.

An argument to the method `main()` is an array of strings containing command line arguments. For example when we use

```
$ java MyClass Hello World
```

in the method `main()` in the class `MyClass` we have

```
args.length = 2
args[0] = "Hello"
args[1] = "World"
```

Program Exit

The program exits when the method `main()` is finished.

> **Note:** In fact, it is more complicated. Generally, program exits when all threads are finished.

It is also possible to use the method `exit()` in the class **java.lang.System** to exit program:

```
System.exit(0);
```

The argument of `exit()` is a a status code. By convention, a nonzero status code indicates abnormal termination.

```
System.exit(1);  // an error occured
```