Variable $\approx$ Box

Variable Declaration

- You give a name for the variable, say x.
- Additionally, you need to assign a type for the variable.
- For example,

```
1 ...
2         int x; // x is a variable declared an interger type.
3 ...
```

- Variable declaration tells the compiler to allocate appropriate memory space for the variable based on its data type.[1]
- It is worth to mention that, the date type determines the size, which is measured in bytes[2].

[1] Actually, all declared variables are created at the compile time.

[2] 1 byte = 8 bits; bit = binary digit.

Naming Rules

- Identifiers are the names that identify the elements such as variables, methods, and classes in the program.
- The naming rule excludes the following situations:
  - cannot start with a digit
  - cannot be any reserved word[3]
  - cannot include any blank between letters
  - cannot contain $+, -, *, /$ and %
- Note that Java is case sensitive[4].

[3] See the next page.

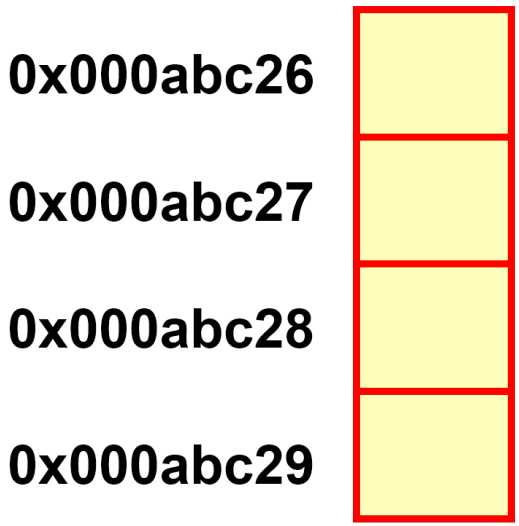
[4] The letter A and a are different.

Reserved Words[5]

abstract	double	int	super
assert	else	interface	switch
boolean	enum	long	synchronized
break	extends	native	this
byte	final	new	throw
case	finally	package	throws
catch	float	private	transient
char	for	protected	try
class	goto	public	void
const	if	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp*	

[5] See Appendix A in YDL, p. 1253.

Variable as Alias of Memory Address



- The number 0x000abc26 stands for one memory address in hexadecimal (0-9, and a-f).[6]
- The variable x itself refers to 0x000abc26 in the program after compilation.

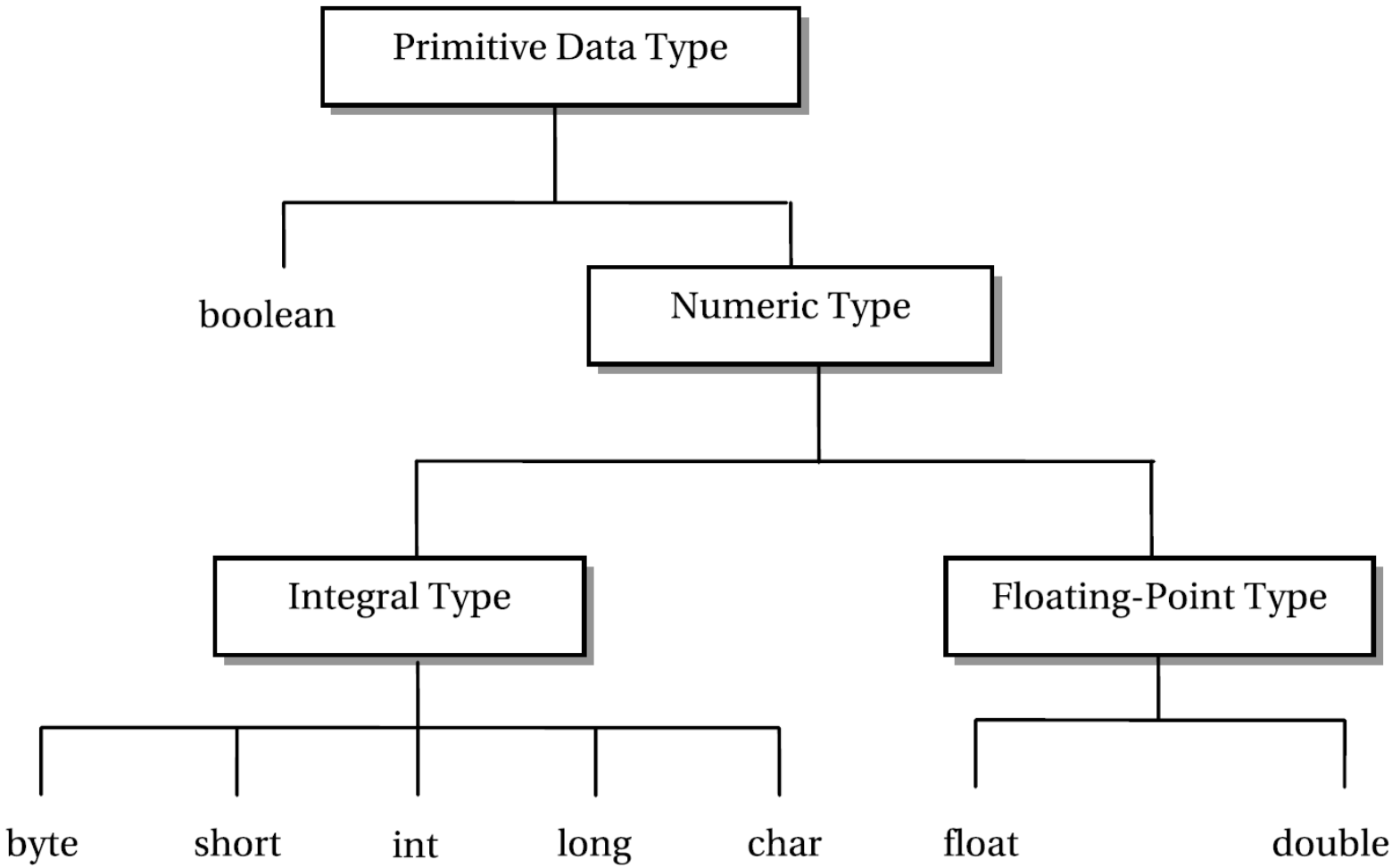
[6] See `https://en.wikipedia.org/wiki/Hexadecimal`.

Data Types

- Java is a static typed[7] programming language.
- Every variable has a type.
- Also, every (mathematical) expression has a type.
- There are two categories of data types: primitive data types, and reference data types.

[7] You cannot change the type of the variable after declaration.

Primitive Data Types[8]



[8] See Figure 3-4 in Sharan, p. 67.

Integers

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

- The most commonly used integer type is int.
- If the integer values are larger than its feasible range, then an overflow occurs.

Floats

Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

- Floats are used when evaluating expressions that require fractional precision.
  - For example, sin(), cos(), and sqrt().
- The performance for the double values is actually faster than that for float values on modern processors that have been optimized for high-speed mathematical calculations.
- Be aware that floating-point arithmetic can only approximate real arithmetic.[9] (Why?)

[9] See `https://en.wikipedia.org/wiki/Numerical_error` and `https://0.30000000000000004.com/`.

Example: Machine Error

```
1 public class NumericalErrorDemo {
2     public static void main(String[] args) {
3
4         System.out.println(0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
5         // output?
6
7     }
8 }
```

- Surprising!!! (Why?)
- Try the decimal-binary converter.
- This issue occurs not only in decimal numbers, but also big integers represented in floats.[10]
- So the floats are not reliable unless the algorithm is designed elaborately for numerical errors.[11]

[10] Thanks to a lively discussion on June 26, 2016.

[11] See https://finance.technews.tw/2017/01/10/largan-stock-trouble/.

Another Example

```
1 ...
2         System.out.println(3.14 + 1e20 - 1e20); // output ?
3         System.out.println(3.14 + (1e20 - 1e20)); // output ?
4 ...
```

- Can you explain why?
- Read this article: What Every Computer Scientist Should Know About Floating-Point Arithmetic.

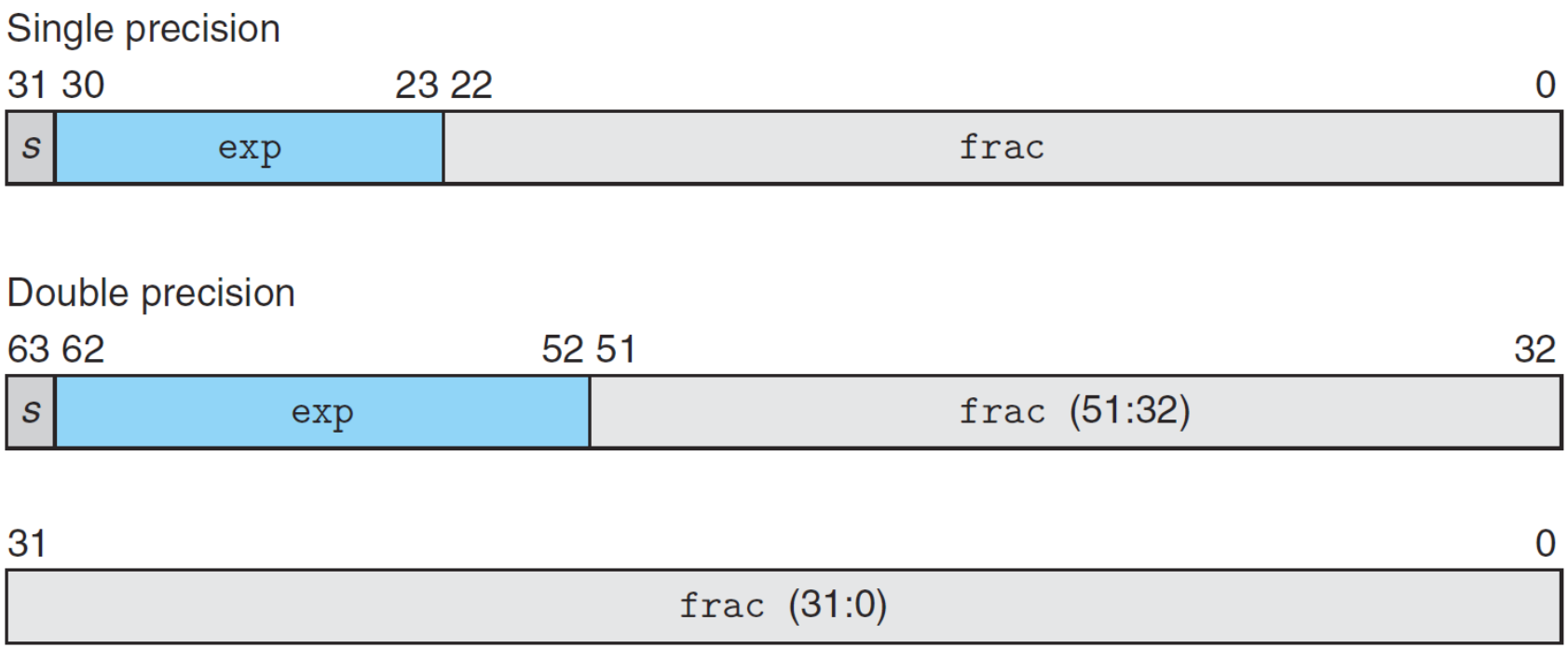
IEEE Floating-Point Representation[12]

$$x = (-1)^s \times M \times 2^E$$

- The sign bit s determines whether the number is negative ($s = 1$) or positive ($s = 0$).
- The mantissa M is a fractional binary number that ranges either between 1 and $2 - \epsilon$, or between 0 and $1 - \epsilon$.
- The exponent E weights the value by a (possibly negative) power of 2.

[12] IEEE754 by William Kahan (1985).

Illustration[13]



- That is why we call a double value.

[13] See Figure 2-31 in Byrant, p. 104.

Assignments

- An assignment statement designates a value to the variable.

```
1        int x; // make a variable declaration
2        ...
3        x = 1; // assign 1 to x
```

- The equal sign (=) is used as the assignment operator.
  - For example, is the expression $x = x + 1$ correct?
  - Direction: from the right-hand side to the left-hand side
- To assign a value to a variable, you must place the variable name to the left of the assignment operator.[14]
  - For example, $1 = x$ is wrong.
  - 1 cannot be resolved to a memory space.

[14] x can be a l-value and r-value, but 1 and other numbers can be only r-value but not l-value. See Value.

Two "Before" Rules

- Every variable has a scope.
  - The scope of a variable is the range of the program where the variable can be referenced.[15]
- A variable must be declared before it can be assigned a value.
  - In practice, do not declare the variable until you need it.
- A declared variable must be assigned a value before it can be used.[16]

[15] The detail of variable scope is introduced later.

[16] In symbolic programming, such as Mathematica and Maple, a variable can be manipulated without assigning a value. For example, $x + x$ returns $2x$.

Arithmetic Operators[17]

Name	*Meaning*	*Example*	*Result*
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

- Note that the operator depends on the operands involved.

[17] See Table 2-3 in YDL, p. 46.

Tricky Pitfalls

- Can you explain this result?

```
1 ...
2         double x = 1 / 2;
3         System.out.println(x); // output?
4 ...
```

- Revisit $0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 = 0$.[18]

```
1 ...
2         System.out.println(1 / 2 - 1 / 10 - 1 / 10 - 1 / 10 - 1
              / 10 - 1 / 10); // output 0; however, this is not
              the real solution to the original problem.
3 ...
```

[18] Thanks to a lively discussion on on June 7, 2016.

Type Conversion and Compatibility

- If a type is compatible to another, then the compiler will perform the conversion implicitly.
  - For example, the integer 1 is compatible to a double value 1.0.
- However, there is no automatic conversion from double to int. (Why?)
- To do so, you must use a **cast**, which performs an explicit conversion for compilation.
- Similarly, a long value is not compatible to int.

Casting

```
1 ...
2         int x = 1;
3         double y = x; // compatible; implicit conversion
4         x = y; // incompatible; need an explicit conversion by
              casting
5         x = (int) y; // succeed!!
6 ...
```

- Note that the Java compiler does only type-checking but no real execution before compilation.
- In other words, the values of x and y are unknown until they are really executed.

Type Conversion and Compatibility (concluded)

- small-size types $\rightarrow$ large-size types
- small-size types $\nleftarrow$ large-size types (need a cast)
- simple types $\rightarrow$ complicated types
- simple types $\nleftarrow$ complicated types (need a cast)

Characters

- A character stored by the machine is represented by a sequence of 0's and 1's.
  - For example, ASCII code. (See the next page.)
- The char type is a 16-bit unsigned primitive data type.[19]

[19] Java uses Unicode to represent characters. Unicode defines a fully international character set that can represent all of the characters found in all human languages.

ASCII (7-bit version)

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Example

- Characters can also be used as (positive) integers on which you can perform arithmetic operations.[20]
- For example,

```
1 ...
2         char x = 'a';  // single-quoted: a char value
3         System.out.println(x + 1); // output 98!!
4         System.out.println((char)(x + 1)); // output b
5
6         String s = "Java"; // double-quoted: a String object
7 ...
```

- You can imagine that a String object comprises characters equipped with plentiful tools.[21]

[20] See `https://en.wikipedia.org/wiki/Cryptography`.

[21] As an analogy, a molecule (string) consists of atoms (characters).

Boolean Values

- The program is supposed to do decision making by itself, for example, Google Driverless Car.[22]
- To do this, Java has the boolean-type flow controls (selections and iterations).
- This type has only two possible values, true and false.
- Note that a boolean value cannot be cast into a value of another type, nor can a value of another type be cast into a boolean value. (Why?)

[22] See `https://www.google.com/selfdrivingcar/`.

Rational Operators[23]

Java Operator	*Mathematics Symbol*	*Name*
<	<	less than
<=	≤	less than or equal to
>	>	greater than
>=	≥	greater than or equal to
==	=	equal to
!=	≠	not equal to

- These operators take two operands.
- Rational expressions return a boolean value.
- Note that the equality operator is double equality sign (==), not single equality sign (=).

[23] See Table 3-1 in YDL, p. 82.

Example

```
1 ...
2         int x = 2;
3         boolean a = x > 1;
4         boolean b = x < 1;
5         boolean c = x == 1;
6         boolean d = x != 1;
7         boolean e = 1 < x < 3; // sorry?
8 ...
```

- Be aware that e is logically correct but syntactically wrong.
- Usually, the boolean expression consists of a combination of rational expressions.
  - For example, $1 < x < 3$ should be $(1 < x)\&\&(x < 3)$, where && refers to the AND operator.

Logical Operators[24]

Operator	*Name*	*Description*
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion

[24] See Table 3-2 in YDL, p. 102.

Truth Table

- Let X and Y be two Boolean variables.
- Then the truth table for logical operators is as follows:

X	Y	!X	X&&Y	X $\|\|$ Y	X $\wedge$ Y
T	T	F	T	T	F
T	F	F	F	T	T
F	T	T	F	T	T
F	F	T	F	F	F

- Note that the instructions of computers, such as arithmetic operations, are implemented by logic gates.[25]

[25] See any textbook for digital circuit design.

"Logic is the anatomy of thought."

– John Locke (1632–1704)

"This sentence is false."

– anonymous

"I know that I know nothing."

– Plato

(In Apology, Plato relates that Socrates accounts for his seeming wiser than any other person because he does not imagine that he knows what he does not know.)

Arithmetic Compound Assignment Operators

++	Increment
+=	Addition assignment
−=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
−−	Decrement

- Note that these shorthand operators are not available in languages such as Matlab and R.

Example

```
1  ...
2          int x = 1;
3          System.out.println(x); // output 1
4          x = x + 1;
5          System.out.println(x); // output 2
6          x += 2;
7          System.out.println(x); // output 4
8          x++; // equivalent to x += 1 and x = x + 1
9          System.out.println(x); // output 5
10 ...
```

- The compound assignment operators are also useful for char values.[26]
- For example,

```
1 ...
2         char s = 'a';
3         System.out.println(s); // output a
4         s += 1;
5         System.out.println(s); // output b
6         s++;
7         System.out.println(s); // output c
8 ...
```

[26] Contribution by Mr. Edward Wang (Java265) on May 1, 2016.

++x vs. x++

- The expression ++x first increments the value of x and then returns x.
- Instead, the expression x++ first returns the value of x and then increments itself.
- For example,

```
1  ...
2          int x = 1;
3          int y = ++x;
4          System.out.println(y); // output 2; aka preincrement
5          System.out.println(x); // output 2
6
7          int w = 1;
8          int z = w++;
9          System.out.println(z); // output 1; aka postincrement
10         System.out.println(w); // output 2
11 ...
```

- We will use these notations very often.

Operator Precedence[27]

Precedence	*Operator*
↓	`var++` and `var--` (Postfix)
	`+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix)
	(type) (Casting)
	`!` (Not)
	`*`, `/`, `%` (Multiplication, division, and remainder)
	`+`, `-` (Binary addition and subtraction)
	`<`, `<=`, `>`, `>=` (Comparison)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

[27] See Table3-10 in YDL, p. 116.

Using Parentheses

- Parentheses are used in expressions to change the natural order of precedence among the operators.
- One always evaluates the expression inside of parentheses first.

Scanner Objects

- It is not convenient to modify the source code and recompile it for a different radius.
- Reading from the console enables the program to receive an input from the user.
- A **Scanner** object provides some input methods, say the input received from the keyboard or the files.
- Java uses **System.in** to refer to the standard input device, by default, the keyboard.

Example: Reading Input From The Console

Write a program which receives a number as input, and outputs the area of the circle.

```
1  import java.util.Scanner;
2  ...
3          Scanner input = new Scanner(System.in);
4          System.out.println("Enter r?");
5          // input
6          int r = input.nextInt();
7          // algorithm
8          double area = r * r * 3.14;
9          // output
10         System.out.println(area);
11         input.close();
12 ...
```

- Line 3 is to create a **Scanner** object by the new operator, as an agent between the keyboard and your program.
- Note that all objects are resided in the heap of memory.
- To manipulate this object, its memory address is then assigned to the variable *input* which is allocated in the stack of memory, aka a reference.
- We will discuss more about objects and references later!