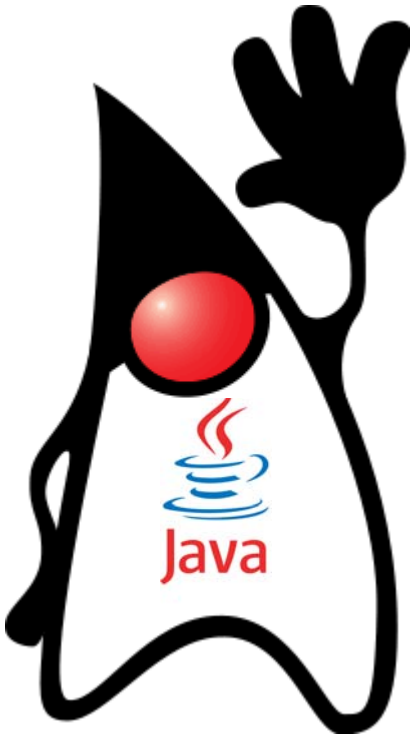
101 Review



Asst. Dr. Umaporn Supasitthimethee

Lexical Structures

- White spaces
- Comments
  - /* text */ /** text */
  - // text
- Tokens
  - Identifiers
  - Keywords / Reserved words
  - Literals
  - Separators
  - Operators

White Spaces

- Spaces, blank lines, and tabs are called white space
- White space is used to separate words and symbols in a program
- Extra white space is ignored
- A valid Java program can be formatted many ways
- Programs should be formatted to enhance readability, using consistent indentation

Java Program

See TestNoFormat1.java

```
public class TestNoFormat1{public static void main(String[]args){
System.out.println("Java");
System.out.println("Compile Once, Run Anywhere");}}
```

See TestNoFormat2.java

```
public       class
     TestNoFormat2   {
                  public static void  main       (
String []                  args )
  {
  System.out.println (
"Java")
  ;         System.out.println
             (
       "Compile Once, Run Anywhere"
      )
  ;
}    }
```

Boths are Bad Layout

Java Program

```
public class TestFormat{
   public static void main(String[]args){
        System.out.println("Java");
        System.out.println("Compile Once, Run Anywhere");
   }
}
```

Good Layout

Comments

- Comments in a program are called *inline documentation*
- Good Comments help programmers to explain the purpose of the program and describe processing steps
- They are not programming statements and thus are ignored by the compiler
- Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks       */

/** this is a javadoc comment   */
```

Identifiers (1)

- Identifiers are the words a programmer uses in a program
- An identifier can be made up of letters, digits, the underscore character (_), and the dollar sign
- Identifiers cannot begin with a digit
- Identifiers cannot be the same as keywords/reserved words
- Java is case sensitive - Total, total, and TOTAL are different identifiers

Identifiers (2)

- By convention, programmers use different case styles for different types of identifiers, such as
  - title case for class names – `BankAccount, Student`
  - upper case for constants – `MAXIMUM`, `TAX`
  - camelCase for variables, methods, attributes, and objects – `studentName,isFull`
- We choose identifiers ourselves when writing a program (such as HelloWorld)
- Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as println)

Keywords/Reserved Words

- The Java reserved words:

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

Primitive Data Types

There are eight primitive data types in Java

- Four of them represent integers:
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers:
  - `float`, `double`
- One of them represents characters:
  - `char`
- One of them represents boolean values:
  - `boolean`

Literals

- *Literal* is a constant value that appears in a program

- **integer**
  - 20 ***(default int)*** 20L 20l
  - 010 (octal) 0Xf0 (hex) 0xffL

- **floating point**
  - 3.14 ***(default double)*** 3.1E12 2.0e-2
  - 2.0F 2.5f

- **character**
  - 'A' '\n' '\udddd' (hex - \udddd)
  - '\\' ' \101' (octal - \ddd)

- **boolean**
  - true
  - false

Separators

( ) Parentheses

- contain lists of parameters in method definition an invocation
- define precedence in expressions
- contain expressions in control statements
- surround cast types

{ } Braces

- contain the values of automatically initialized arrays
- define a block of code for classes, methods, and local scopes

[ ] Brackets

- declare array types
- dereference array values

Separators

; Semicolon

- terminates statements

, Comma

- separates consecutive identifiers in a variable declaration
- use to chain statements together inside a for statement

. Period

- separate package names from subpackages and classes
- use to separate an attribute or method from a reference variable

Operators

- **Arithmetic**

  + - * / %

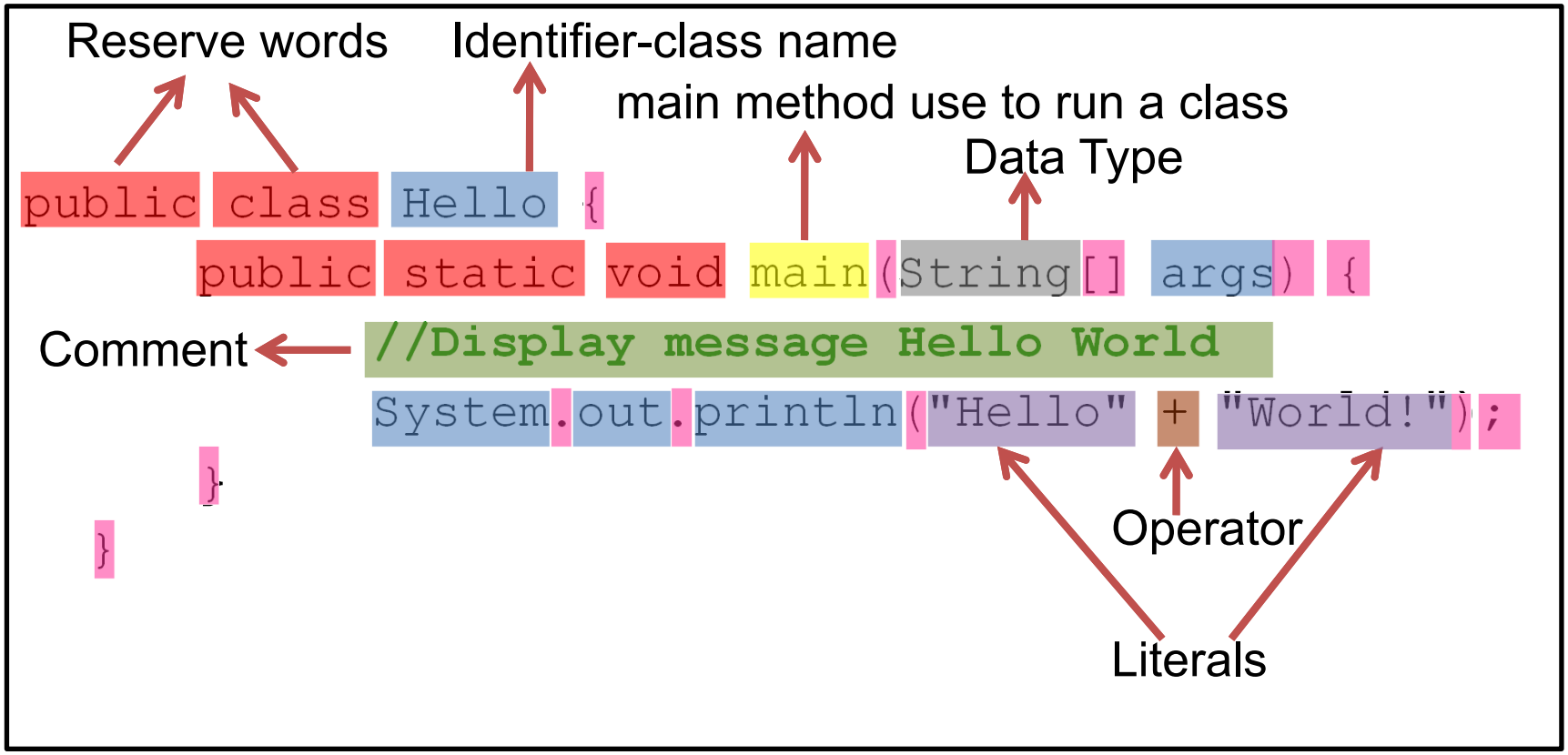
- **Relational**

  < <= > >= == !=

- **Logical**

  ! && & || |

Program Structures



reserve words | main method | literals | separators

identifiers | comment | operator | data type

Primitive Data Type

Type	Size	Default	Value Ranges	Contains
boolean	16 bits	false		true or false
byte	8 bits	0	-128 to 127	Signed integer
char	16 bits	\u0000		Unicode character
short	16 bits	0	-32,768 to 32,767	Signed integer
int	32 bits	0	-2,147,483,648 to 2,147,483,647	Signed integer
long	64 bits	0	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed integer
float	32 bits	0.0	Approximately -3.4E+38 to 3.4E+38 with 7 significant digits	IEEE754 floating point
double	64 bits	0.0	Approximately -1.7E+308 to 1.7E+308 with 15 significant digits	IEEE754 floating point

boolean

- A `boolean` value represents a true or false condition
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

```
boolean done =“true”;
```

```
boolean done  = True;
```

- A `boolean` variable can also be used to represent any two states, such as a light bulb being on or off

Integer

- Four of them represent integers:
  - `byte`, `short`, `int(default)`, `long`
- Example declarations:
  - `byte b=120;`
  - `short s=22112;`
  - `int i=13000;`
  - `long l=2124230;`

Floating Point

- Two of them represent floating point numbers:
  - `float`, `double(default)`

- Example declarations:
  - `float b= 123.23f;`
  - `double d=123.23;`

Character

- A char variable stores a single character by using single quotes:
- `'A' 'b' '\n' '$' ','`

- Example declarations:
  - `char grade = 'A';`
  - `char symbol = ';';`

String

- A string of characters can be represented as a *string literal* by putting double quotes around the text.
- Note the distinction between a primitive character variable, which holds only one character, and a String object, which can hold multiple characters
- Examples:

```
"This is a string example."
"SIT, KMUTT"
"A"
"Java\nProgramming"
```

String Concatenation (1)

- The string concatenation operator (+) is used to append one string to the end of another
- `" Java " + " Programming"`
- It can also be used to append a number to a string
- A string literal cannot be broken across two lines in a program

```
System.out.println ("learning" + 60 + " hours per course");
```

String Concatenation (2)

- The + operator is also used for arithmetic addition
- The function that it performs depends on the type of the information on which it operates
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation
- If both operands are numeric, it adds them
- The + operator is evaluated left to right, but parentheses can be used to force the order

```
System.out.println ("24 and 45 concatenated: " + 24 + 45);
System.out.println ("24 and 45 added: " + (24 + 45));
```

Escape Sequences

- What If we wanted to print the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

- `System.out.println ("I said "Hello" to you.");`

- An escape sequence is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

- `System.out.println ("I said \"Hello\" \nto you.");`

Escape Sequences

- Some Java escape sequences:

Escape Sequence	Name
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

Variables

- A variable is a name for a location in memory
- A variable must be declared by specifying the variable's name and the type of information that it will hold
- You must declare variable before using

data type

variable name

```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

Variable Initializations

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

```
 int point = 40;
System.out.println ("your score is " + point  + " points.");
```

Assignments

- An assignment statement changes the value of a variable
- The assignment operator is the = sign

```
total = 55;
```

- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in total is overwritten
- You can only assign a value to a variable that is consistent with the variable's declared type

Named Constants (1)

- A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence
- As the name implies, it is constant, not variable
- The compiler will issue an error if you try to change the value of a constant
- In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

Named Constants (2)

- Constants are useful for three important reasons
- First, they give meaning to otherwise unclear literal values
  - For example, `MAX_LOAD` means more than the literal 250
- Second, they facilitate program maintenance
  - If a constant is used in multiple places, its value need only be updated in one place
- Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers

Expressions

- An *expression* is a combination of one or more operators and operands
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

```
Addition        +
Subtraction     -
Multiplication  *
Division        /
Remainder       %
```

If either or both operands used by an arithmetic operator are floating point, then the result is a floating point

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3      equals      4

8 / 12      equals      0
```

The remainder operator (%) returns the remainder after dividing the second operand into the first

```
14 % 3      equals      2

8 % 12      equals      8
```

Operator Precedence

- Operators can be combined into complex expressions

```
result = total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order

Precedence Level	Operator	Operation	Associativity
1	Method()	Method Invocation	L to R
	.	Object Member Reference	
	[]	Array Indexing	
	++, --	**Post-Increment, Post-Decrement**	
2	**++, --**	**Pre-Increment, Pre-Decrement**	R to L
	+, -	**Unary Plus , Unary Minus**	
	!	Logical Not	
3	new	Object Instantiation	R to L
	(<type>)	**Cast (Type Conversion)**	
4	***, /, %**	**Arithmetic Operators**	L to R
5	**+, -**	**Arithmetic Operators**	L to R
	+	**String Concatenation**	
6	<, <=, >, >=	Relational Operators	L to R
7	==, !=	Equality Operators	L to R
8	&	Logical AND	L to R
9	|	Logical OR	L to R
10	&&	Short-Circuit AND	L to R
11	||	Short-Circuit OR	L to R
12	?:	Conditional Operator	R to L
13	**=, *=, /=, %=, +=, -=**	**Assignment with operation**	R to L

Order of Evaluation

- When the Java interpreter evaluates an expression, it performs the various operations in an order specified by
  - the parentheses in the expression
  - the precedence of the operators
  - the associativity of the operators.
- Before any operation is performed, however, the interpreter first evaluates the operands of the operator.
- The interpreter always evaluates operands in order from left to right.
- This matters if any of the operands are expressions that contain side effects

Operator Precedence

- What is the order of evaluation in the following expressions?

```
a + b + c + d + e
  1   2   3   4
```

```
a + b * c - d / e
  3   1   4   2
```

```
a / (b + c) - d % e
  2     1     4   3
```

```
a / (b * (c + (d - e)))
  4     3     2     1
```

Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
```

4 1 3 2

Then the result is stored in the variable on the left hand side

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of count

```
count  =  count + 1;
```

Then the result is stored back into count (overwriting the original value)

Increment and Decrement

- The increment and decrement operators use only one operand
- The *increment operator* (++) adds one to its operand
- The *decrement operator* (--) subtracts one from its operand
- The statement

```
count++;
```

  is functionally equivalent to

```
count = count + 1;
```

Increment and Decrement

- The increment and decrement operators can be applied in *postfix form*:

```
count++
```

- or *prefix form*:

```
++count
```

- When used as part of a larger expression, the two forms can have different effects
- Because of their subtleties, the increment and decrement operators should be used with care

Increment and Decrement Example

```
int num1=1;
int num2=1;
num1++;
++num2;
System.out.println(num1); //num1 = 2
System.out.println(num2); //num2 = 2
```

Increment and Decrement Example

```
int num1=1;
int num2=1;

int num3=++num1 + 2;
System.out.println(num3); //num3 = 4
System.out.println(num1); //num1 = 2

int num4=num2++ + 2;
System.out.println(num4); //num4 = 3
System.out.println(num2);//num2 = 2
```

Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

- There are many assignment operators in Java, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```

Assignment Operator Example

```
int result=50, num=4, total=11;
final int MIN=4;
result /= (total-MIN) % num;
System.out.println(result); //result = 16
```

Assignment Operators

- The behavior of some assignment operators depends on the types of the operands
- If the operands to the += operator are strings, the assignment operator performs string concatenation
- The behavior of an assignment operator (+=) is always consistent with the behavior of the corresponding operator (+)

Type Conversions

- Java is strongly typed checking, that means each data value is associated with a particular type.
- It is sometimes helpful or necessary to convert a data value of one type to another type, but we must be careful that we don't lose important information in the process
- There is strict enforcement of type rules
  - **Widening Conversions**
  - **Narrowing Conversions**
- Note that Widening a type can be performed automatically without explicit casting. Narrowing a type must be performed explicitly

Type Conversions (2)

- **Widening Conversions** – are the safest because they usually do not lose information. They convert from one data type to another type that uses greater amount of space to store the value (such as a `short` to an `int`)
- **Narrowing Conversions** – are more likely to lose information than widening conversions are. They often convert from one type to a type that use less space to store a value, and therefore some of the information may be lost (such as an `int` to a `short`)
- These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation

Java Widening Conversions

Type	Convert to
byte	short, int, long, float, double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

Note that When converting int or long to float or from long to double, some of the significant digits may be lost precision

Java Narrowing Conversions

Type	Convert to
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

Note that boolean values cannot be converted to any other primitive type and vice versa

Data Conversions

- In Java, data conversions can occur in three ways:
  - Assignment conversion
  - Numeric promotion
  - Casting conversion

Assignment Conversions

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
- If `money` is a `float` variable and `dollars` is an `int` variable, the following assignment converts the value in `dollars` to a `float`

```
money = dollars
```

- Only widening conversions can happen via assignment
- Note that the value or type of `dollars` did not change

Numeric Promotions

- *Numeric Promotion* happens automatically when operators in expressions convert their operands
- For example, if `sum` is a `float` and `count` is an `int`, the value of `count` is converted to a floating point value to perform the following calculation:

```
result = sum / count;
```

Casting Conversions

- *Casting conversion* is the most powerful, and dangerous, technique for conversion
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted
- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:

```
result = (float) total / count;
```

Interactive Programs

- Programs generally need input on which to operate
- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

Reading Input

- The following line creates a Scanner object that reads from the keyboard:

```
Scanner scan = new Scanner (System.in);
```

- The `new` operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
answer = scan.nextLine();
```

Reading Input

- The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used
- The `nextLine` method reads all of the input until the end of the line is found
- Unless specified otherwise, white space is used to separate the elements (called tokens) of the input
- White space includes space characters, tabs, new line characters
- The `next` method of the Scanner class reads the next input token and returns it as a string
- Methods such as `nextBoolean`, `nextByte`, `nextShort`, `nextInt`, `nextLong`, `nextFloat` and `nextDouble` read data of particular types

Scanner Example

```
Scanner sc = new Scanner(System.in);

boolean bo = sc.nextBoolean();

byte b = sc.nextByte();
short s = sc.nextShort();
int x = sc.nextInt();
long l = sc.nextLong();

double a = sc.nextDouble();
float f = sc.nextFloat();

String st1= sc.nextLine();
//the rest of the current line,  excluding any line separator at the end
String st2= sc.next();
//returns the next complete token
```

The Math Class

- The `Math` class is part of the `java.lang` package
- The `Math` class contains methods that perform various mathematical functions
- These include:
  - absolute value
  - square root
  - exponentiation
  - trigonometric functions

Math Methods

- `public static double pow(double a, double b)`
- `public static double random()`
- `public static double sqrt(double a)`
- `public static double max(double a, double b)`
- `public static float max(float a, float b)`
- `public static long max(long a, long b)`
- `public static int max(int a, int b)`
- `public static double min(double a, double b)`
- `public static float min(float a, float b)`
- `public static long min(long a, long b)`
- `public static int min(int a, int b)`

Math.random()

- public static double **random**()

  // Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

- Random number in the range (n_1-n_2)

  $(int)(Math.random() * (n_2 - n_1 + 1)) + n_1$

- For example,
- Random the number in the range 1-6

```
int i=(int)(Math.random()*6)+1;
```

Math Constant

- `public static final double` **`PI`**

- For example,

```
System.out.println(Math.PI);

//3.141592653589793
```

The Random Class

- The `Random` class is part of the `java.util` package
- It provides methods that generate pseudorandom numbers
- A `Random` object performs complicated calculations based on a *seed value* to produce a stream of seemingly random values
  - `public int nextInt(int n)`
  - `public int nextInt()`
  - `public double nextDouble() //same as Math.random()`
  - `public float nextFloat()`

The Random Class

- Random number in the range (n_1-n_2)

```
randObj.nextInt(n2 – n1 + 1) + n1
```

- For example,
- Random the number in the range 20-34

```
Random generator = new Random();
int num1 = generator.nextInt(15) + 20;
```

Object Oriented Concepts

- ADT: Abstract Data Type
- Encapsulation
- Class & Object
  - Methods/Operations/Functions
  - Attributes/Instance data

ADT: Abstract Data Type

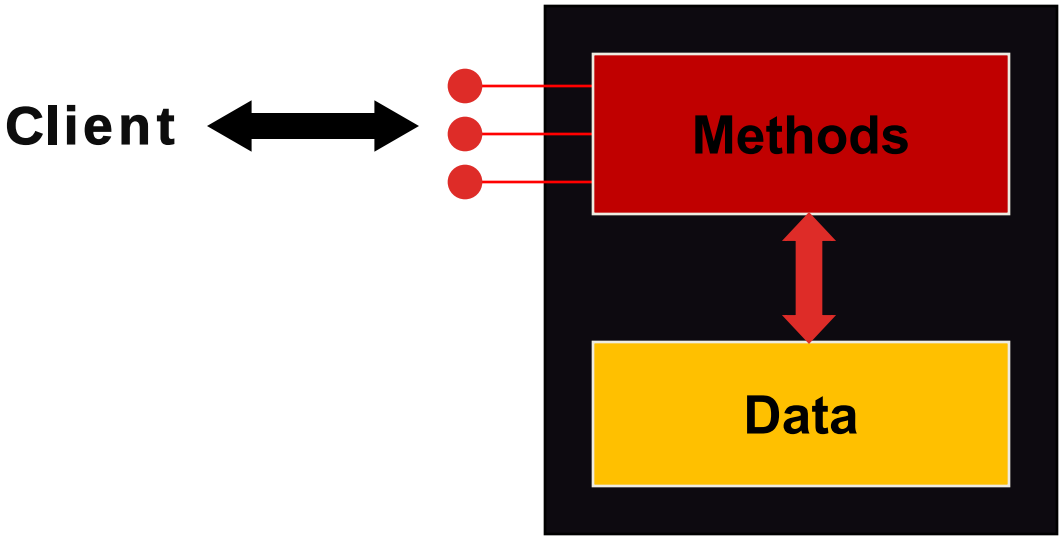
- Abstract Data Type (ADT) is a collection of data and the particular operations that are allowed on that data.
- ADT whose data representation is hidden with private access modifier and define interface as operations having public access modifier.
- There are two parts
  - an element of ADT data
    - Attributes
  - an implementation of ADT operation
    - Methods

Encapsulation

- We can take one of two views of an object:
  - internal - the details of the variables and methods of the class that defines it
  - external - the services that an object provides and how the object interacts with the rest of the system
- From the external view, an object is an *encapsulated* entity, providing a set of specific services
- These services define the *interface* to the object

Encapsulation

- An encapsulated object can be thought of as a black box -- its inner workings are hidden from the client
- The client invokes the interface methods of the object, which manages the instance data



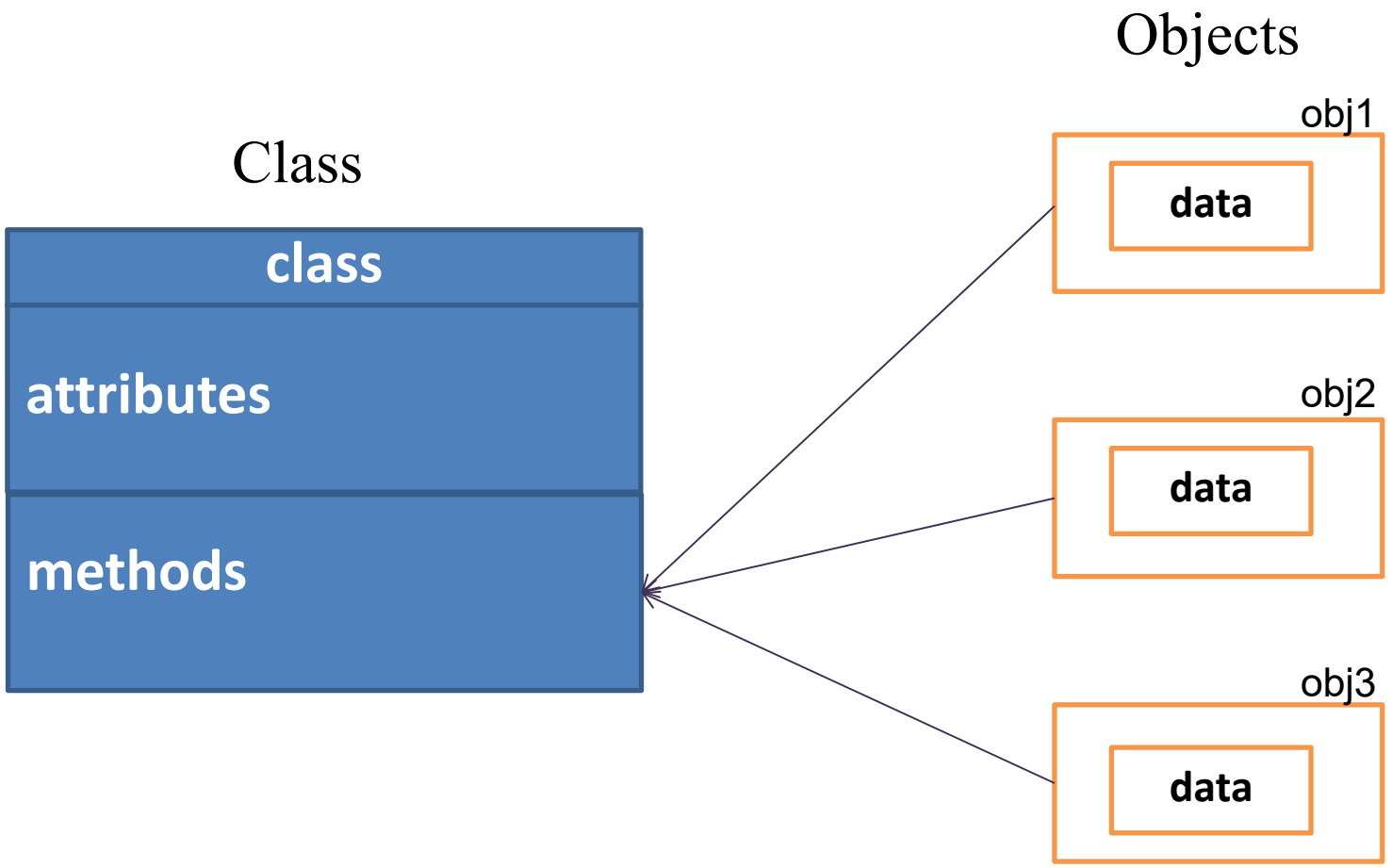
Encapsulation

- The process shielding an object's data from direct outside access.
- Objects have no direct access to the data in other objects.
- Only the operations that are included in an object directly access the object's own data.
- Data is said to be hidden because access is only possible indirectly.

Class & Object

- A class is a template or blueprint for creating an object.
- A class defines what types of data are included in the object and specifies the operations the object performs.
- Object is created from a class.
- Creating an object from a class is known as instantiating an object.

Class & Object



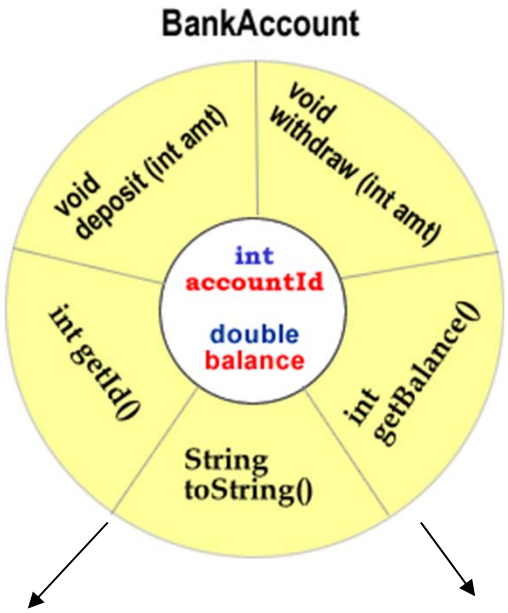
Bank Account

- an account number
- an account owner
- a balance
- a transaction history

- deposit – an amount to deposit, the date of deposit
- withdraw – an amount to withdraw, the date of withdraw
- transfer – an amount to transfer, an account to transfer to, the date of transfer
- inquiry – (the date of inquiry)

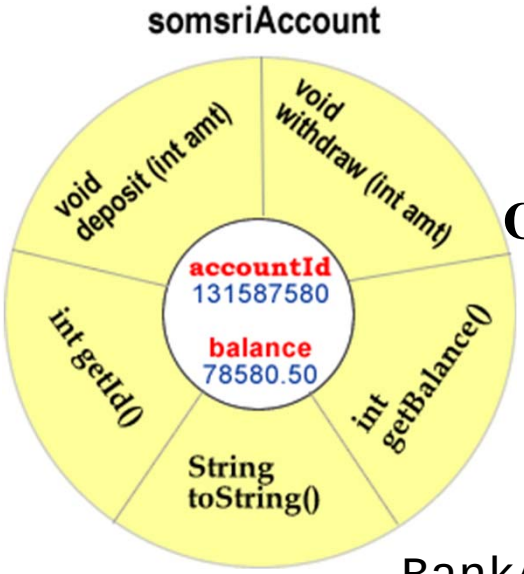
- adding an interest – the date of adding the interest, (interest rate)

- open a new account – an account owner, the date of account opening
- close the account – the date of account closing

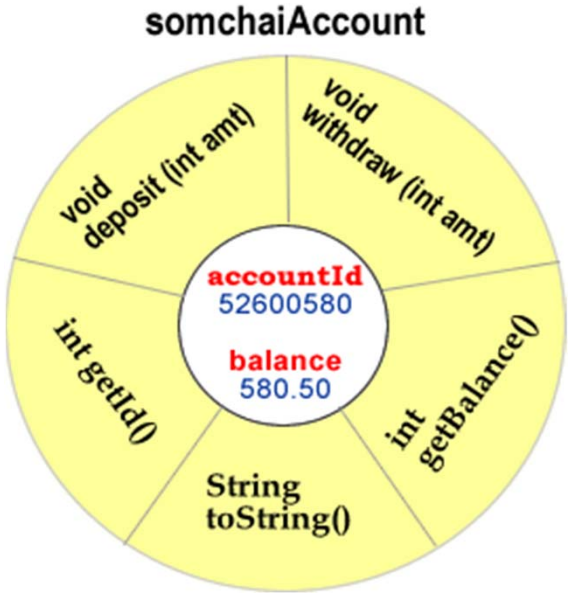


Class

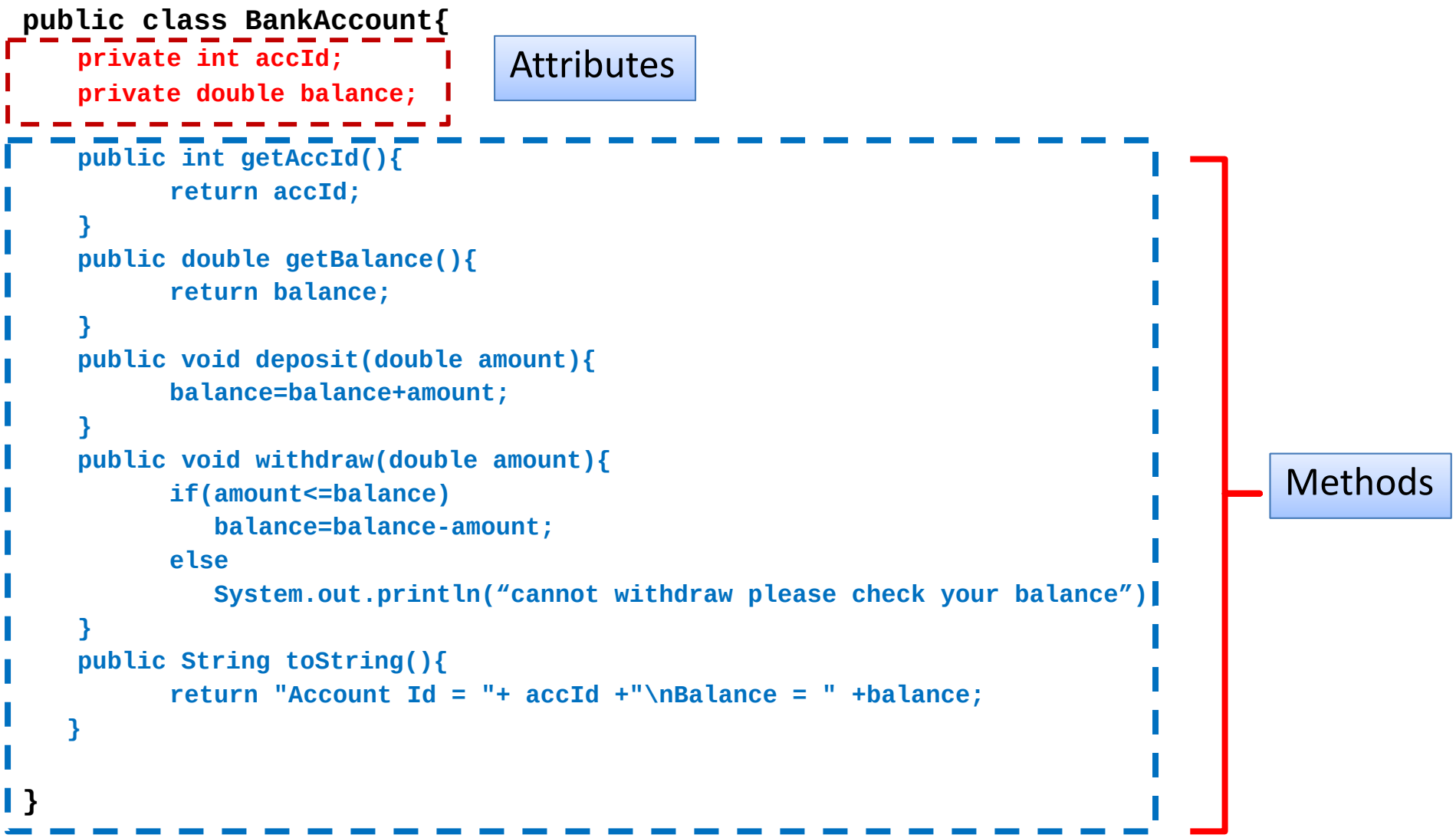
```
BankAccount somsriAccount;
somsriAccount = new BankAccount();
```



Objects/Instances




```
BankAccount somchaiAccount= new BankAccount();
```


public class BankAccount{
private int accId;
private double balance;
Attributes
public int getAccId(){
return accId;
}
public double getBalance(){
return balance;
}
public void deposit(double amount){
balance=balance+amount;
}
public void withdraw(double amount){
if(amount<=balance)
balance=balance-amount;
else
System.out.println("cannot withdraw please check your balance")
}
public String toString(){
return "Account Id = "+ accId +"\nBalance = " +balance;
}
}
Methods

Write a program to find out the Chinese Zodiac sign for a given year. The Chinese Zodiac is based on a twelve-year cycle, with each year represented by an animal-monkey, rooster, dog, pig, rat, ox, tiger, rabbit, dragon, snake, horse, or sheep-in this cycle, as shown in. Note that **year % 12** determines the Zodiac sign. 1900 is the year of the rat because **1900% 12** is **4**. Write a program that prompts the user to enter a year and displays the animal for the year.

year % 12 =
- 0: monkey
- 1: rooster
- 2: dog
- 3: pig
- 4: rat
- 5: ox
- 6: tiger
- 7: rabbit
- 8: dragon
- 9: snake
- 10: horse
- 11: sheep

```
Enter a year: 1963 Enter
rabbit
```

```
Enter a year: 1877 Enter
ox
```

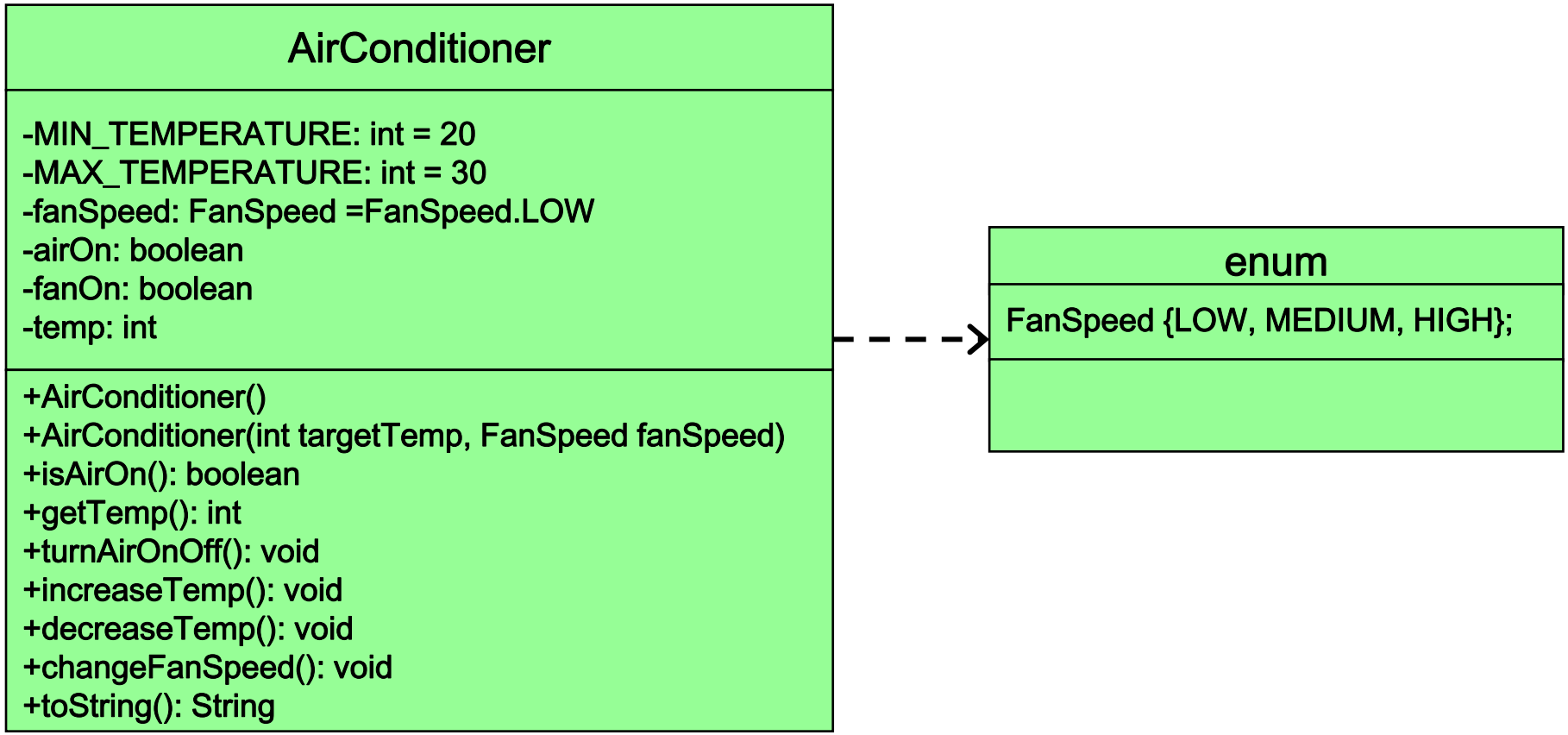
Air Conditioner

- turn on
- turn off
- increase/decrease temperature
- increase/decrease fan speed

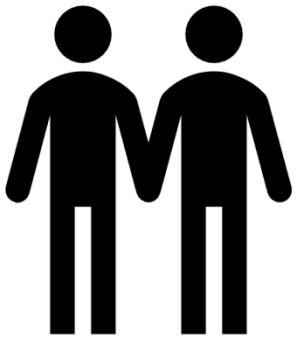
Requirement

- When starting the air conditioner, the air conditioner and fan will be on and the temperature starting at 25 (C°) and Low fan speed.
- The `target temperature` will range between `20` and `30` Celsius(C°).
- The `fan speed` can be “LOW”, “MEDIUM” or “HIGH”.
- The `changeFanSpeed()` will increase the fan speed by one. If the fan speed is greater than the maximum fan speed, the fan speed is set to the minimum fan speed.
- The `turnAirOnOff()` method when the current status of air conditioner is on the air conditioner will be set both `airOn` and `fanOn` attributes to `false`. When the current status of air conditioner is off, it will turn on by setting both `airOn` and `fanOn` attributes to `true`.
- The `isAirOn()` method will return boolean value. It returns `true` when the air condition is on. Otherwise, it returns `false`.
- The `increaseTemperature()` will increase the target temperature by one. If the target temperature is greater than the maximum temperature, the target temperature is set to the maximum temperature.
- The `decreaseTemperature()` will decrease the target temperature by one. If the target temperature is lower than the minimum temperature, the target temperature is set to the minimum temperature.
- The `getTemperature()` method will return temperature of air conditioner in Celsius(C°).

Air Conditioner



Design and Develop your rolling Dice Game



2 Players

UML Class Diagrams