AP CS Notes

William Huang

No Institute Given

Abstract. Notes on AP CS 2021. May not cover all the material for different years.

Keywords: AP Computer Science · Computer Science

1 Intro Java Language Features

1.1 Basics

Definition 1. *A class is an object constructor, grouped into packages like* `java.util`

Import statements have the format `import packagename.subpackagename.ClassName;`

Definition 2. *A compiler converts source code into machine-readable bytecode.*

Definition 3. *Reserved words or keywords (e.g.* `void`*) have specific uses in Java, and may not be used as identifiers*

Definition 4. *The keyword* `static` *is used for methods that will not access any objects of a class, typically true for a source file with no instance variables. The* `main` *method must be static.*

Types of comments:

1. `/*...*/` encloses a block
2. `//` comments one line
3. `/**...*/` generates Javadoc comments.

Definition 5. *An identifier is a name for a variable, parameter, etc. Any letters, digits, and underscore. May not begin with digit.*

Built-in types

1. `int`: integer
2. `boolean`: true or false
3. `double`: a double precision floating-point number

Remark 1. If you have too big an `int`, you will get an overflow error, but no warning just a wrong result.

If compatible types, can be cast, e.g.:
```
int x;
double y;
y = (double) x;
```

Remark 2. Casting double to int truncates the number. Note that `(int) (doubleVar - 0.5)` correctly rounds in all cases.

Integers have 32 bits (4 bytes), using one bit for sign, so:

1. `Integer.MAX_VALUE` $= 2^{31} - 1$
2. `Integer.MIN_VALUE` $= -2^{31}$ (since 0 does not need to be stored twice)

Remark 3. Floating-point numbers have the following errors (none throw exceptions):

1. round-off error, since floating-point numbers cannot be represented exactly
2. undefined results, such as `0.0/0.0` return `NaN`.
3. infinitely large or small numbers, like `3.0/0.0`, return `Infinity` or `-Infinity`

Definition 6. *A final variable or user-defined constant will not change value.*

Example 1. Final variable declarations:
```
final double TAX = 0.08;
final double TAX_RATE;
if( <some condition> )
   TAX_RATE = 0.08;
else
   TAX_RATE = 0.0;
```

1.2 Operators

Arithmetic Operators: +, -, *, /, % (mod).

Remark 4. Note:

1. For any operation involving a double and int, the int is promoted to a double.
2. Mod works on doubles too, e.g. `4.2 % 2.0` gives `0.2`.
3. Integer division truncates the answer, e.g. `3/4` gives `0`

Relational Operators: ==, !=, <, >, <=, >=

Remark 5. For floating point numbers, you should code:

$$|x - y| \leq \varepsilon \max(|x|, |y|)$$

where ε is typically 10^{-16}

Logical/Boolean Operators: !, &&, ||

Remark 6. Short circuit evaluation: boolean expressions are evaluated left to right, and evaluation automatically stops if value of entire expression is known.

Assignment Operators: =, +=, -=, *=, /=, %= (all but = sometimes called compound assignment operators)

Increment, Decrement: ++, −−

Operator Precedence:

1. !, ++, −−
2. *, /, %
3. +, -
4. <, >, <=, >=
5. ==, !=
6. &&
7. ||
8. =, *=, -=, *=, /=, %=

Acronym: [NI]MILO [Not/Increment], Math, Inequalities, Logic, Operators.

1.3 More Basics

Input/Output: `System.out.print` and `System.out.println`

Escape Sequences: \n, \", \\

Control Structures: `if... else if ... else`

Remark 7. Note: `else` is always matched with nearest `if`, e.g.

```
if(n > 0)
   if(n % 2 == 0)
      do something
else
   do something else
```

In this example, if n is 3, `do something else` will occur, since `else` is attached to `if(n%2 == 0)`. Use brackets { } to correct this.

Iteration: `for`, `while`

Example 2. An example for loop is:

```
for (int j=2; j <= 10; j++)
{
   statements;
}
```

An example while loop is:
```
int j = 2;
while(j <= 10)
{
   j++;
}
```

Enhanced for loop/for-each loop:
```
for(SomeType element : collection)
{
   statements
}
```

Remark 8. Note: it is possible for the body of a while loop to never be executed, if the test evaluates to `false` the first time.

Errors and Exceptions:

1. `ArithmeticException`: e.g. dividing by 0.
2. `NullPointerException`: any method call on the `null` reference.
3. `ArrayIndexOutOfBoundsException`: out of bounds array index.
4. `IndexOutOfBoundsException`: when `ArrayList` index is out of bounds.
5. `IllegalArgumentException`: when the objects passed into a method are of the wrong type
6. `ConcurrentModificationException`: trying to add or delete elements during enhanced for loop.

2 Classes and Objects

2.1 Definitions

Definition 7. *An object is something that is created an manipulated by a program, and has operations that manipulate itself and other things.*

Definition 8. *The state, or attributes, of an object consists of its data fields/instance variables. The behavior of an object is what it is doing, given by the methods.*

Definition 9. *A class is a software blueprint for implementing objects of a type. An object is a single instance of the class.*

Definition 10. *Combining data and methods into a single class is data encapsulation.*

Definition 11. *Client programs are pieces of code outside a certain class and use that class.*

Public, Private, Static

1. *public* means accessible to all client programs
2. *private* means accessed only by the methods of that class
3. *static* means:
   (a) a variable that is shared by all instances
   (b) a method that performs operations for the entire class, and does not use instance variables of a single instance.

2.2 Methods

Methods, e.g. `public void withdraw(String password, double amount)`
In order, they are the *access specifier, return type, method name,* and *parameter list.*

Types:

1. *constructors*: create an object of the class. Has same name as class, has no return type.
2. *accessors*: returns information without altering anything.
3. *mutators*: changes the state of an object, often `void` type.
4. all the methods above are *instance methods*, as opposed to *static methods.*

Method calling:

Example 3. Example instance method:
```
b.deposit(acctPassword, amount);
```

Example static method:
```
BankAccount.getInterestRate();
```

Definition 12. *An overloaded method: two or more methods with same name but different parameter lists.*

Definition 13. *A method's signature is the method's name and a list of parameter types.*

Definition 14. *The scope of a variable or method is the region it is visible and can be accessed in.*

Remark 9. Local variables take precedence over instance variables of the same name.

`this` object is an implicit parameter for every instance method.

2.3 References

Definition 15. *`double`, `int`, `char`, `boolean` are primitive data types, while all objects and arrays are reference data types.*

Definition 16. *Aliasing is having two references for the same object. This usually is unintended and causes problems.*

Example 4. In the following example, changing a changes b:
```
Date a = new Date(1, 2, 2002);
Date b = a;
```

Example 5. In this example, changing a does not change b:
```
Date a = new Date(1, 2, 2002);
Date b = new Date(1, 2, 2002);
```

Definition 17. *An uninitialized object variable is a null reference or null pointer.*

Dummy/formal parameters are best contrasted with *actual parameters* with an example:

Example 6. An example class:
```
public class BankAccount
{ ...
   public void withdraw(String pass, double amount)
      ...
```

Example usage of class:
```
BankAccount b = new BankAccount("TimB", 1000);
b.withdraw("TimB", 250);
```

In the above example, `pass` and `amount` are the dummy variables while `"TimB"` and `250` are the actual parameters.

Remark 10. **IMPORTANT:** primitives are passed by value, while objects are passed by reference. Sometimes, you will hear that "object references are passed by value" but that's literally objects passed by reference.

3 Inheritance and Polymorphism

3.1 Superclasses and Subclasses

Definition 18. *Inheritance is the relationship between objects that share characteristics.*

Definition 19. *A subclass is a more specific version of a class created from a superclass. The subclass inherits characteristics from its superclass, sometimes denoted as "a [subclass]* is-a *[superclass]"*

Definition 20. *Method overriding is when a subclass redefines a method it inherits. If part of the original implentation is retained, it is called partial overriding.*

Example 7. `public class Subclass extends Superclass`

Remark 11. A subclass does not inherit private instance variables or private methods. To access private instance variables, subclass must use accessor, mutator methods. Private methods cannot be overwritten.

Example 8. Let's say that the superclass has a method `method(int num)`. Any public method can be overridden by defining a method with same return type and signature:
```
public int method(int num){
   int a = super.method(num);
```

```
    return a + num;
}
```

In this case, *partial overriding* occurred by calling on the method of the superclass. Note that `super.method()` can be used in methods beyond `method()`.

Constructors are never inherited:

1. If no constructor in subclass, superclass constructor with no parameters is used, i.e.
   `public SubClass() { super(); }`
2. If the superclass has no zero-parameter constructor, a compiler error occurs.
3. Instance variables in the subclass get default values: 0 for primitives and `null` for reference types.

Remark 12. If `super()` is used in the implementation of a subclass constructor, it must be used in the first line.

Superclass declaration can refer to a subclass object, e.g.
`SuperClass s = new SubClass();`

3.2 Polymorphism

Definition 21. *A method that has been overridden in at least one subclass is polymorphic. Polymorphism is the mechanism of selecting the appropriate method.*

Definition 22. *A run-time decision on which instance method to call is dynamic/late binding. Selecting the correct method at compile time is static/early binding.*

The compiler determines if a method can be called (does it have the right signature), and the run-time environment determines how it will be called (which overridden form to use).

In Java, for overridden methods, the method of the subclass will be called because of dynamic binding.

Remark 13. Note that methods that are only present in subclasses (and not the superclass) will not be able to be called in constructions like `SuperClass s = new SubClass();`. In these situations, you must downcast, e.g.
`int x = ((SubClass) s).getID();`

Remark 14. The dot operator has a higher precedence than casting, so the outer parenthesis are necessary when downcasting.

4 Some Standard Classes

4.1 `Object` Class

Every class automatically extends `Object`. Methods:

1. `public String toString()`: called when trying to print an object. Default method prints the address of the memory, e.g. `Object@fea485c4`.
2. `public boolean equals(Object other)`: returns true if same object, false if not. By default, checks if references are same, equivalent to `==`.

4.2 `String` class

A string literal consists of zero or more characters, including escape sequences, surrounded by double quotes. String objects are *immutable*.

Remark 15. Strings are unusual in that you can initialize it like a primitive type:
```
String a = "abc";
String a = new String("abc");
```
The above two lines are equivalent.

Definition 23. *+ is the concatenation operator for strings. It produces a single string with the left hand string followed by the right hand string.*

Remark 16. Notes about the concatenation operator:

1. If one side is not a string, the `toString()` method for objects is used and concatenation occurs
2. If neither side is a string, an error occurs

Methods:

1. `equals(Object other)`: overridden so that if identical strings (regardless of references), returns true
2. `int compareTo(String otherString)` returns a negative integer if `string` is before `otherString` and vice versa, and returns 0 if identical.
3. `int length()` returns the length of this string
4. `String substring(int startIndex)`, `(int startIndex, int endIndex)`: returns new string that starts at `startIndex` and ends at `endIndex`-1 inclusive or the end of the string. Throws `StringIndexOutOfBoundsException` if indexes are off, but on AP Exam will say `IndexOutOfBoundsException`.
5. `int indexOf(String str)`: returns index of first occurrence of `str` in the string, and if it is not within the string returns -1. Throws `NullPointerException` if `str` is `null`.

Remark 17. `compareTo` uses the ASCII chart, in which digits precede capital letters, which precede lowercase letters.

4.3 Other Classes

Definition 24. *Wrapper classes take an existing object or primitive type, "wrap" or "boxes" it in an object, and provides a new set of methods for that type.*

`Integer` **Class**: contains just one `int` instance variable. Methods:

1. `Integer(int value)`: constructs `Integer` from `int` (boxing).
2. `int intValue()`: returns value of `int` (unboxing).
3. `Integer.MIN_VALUE`: constant equal to minimum value represented by `int` or `Integer`, -2^{31}.
4. `Integer.MAX_VALUE`: constant equal to maximum value represented by `int` or `Integer`, $2^{31} - 1$.

`Double` **Class**: contains just one `double` instance variable. Methods:

1. `Double(double value)`: constructs `Double` from `double` (boxing).
2. `double doubleValue()`: returns value of `double` (unboxing).

Remark 18. Notes:

1. `Integer` and `Double` objects are immutable.
2. Autoboxing occurs when primitive value assigned to wrapper class, e.g. `Integer a = 3;`, or when primitive value passed as parameter when expected class is wrapper, e.g. `list.add(4);` when `list` is an `ArrayList<Integer>`, or when a comparison between `Integer` and `int` occurs, e.g. `intObj < 3`.
3. Auto-unboxing occurs for the same situations as above, with types reversed (except the comparison).
4. Don't compare `Integer` objects with `==`, compare with `.intValue()`

`Math` **Class**: standard math functions. Methods:

1. `static int abs(int x)`: returns absolute value
2. `static double abs(double x)`: returns absolute value
3. `static double pow(double base, double exp)`: returns base^{exp}, assumes base > 0, bass $= 0$ and exp > 0, or base < 0 and exp is an integer.
4. `static double sqrt(double x)`: returns $\sqrt{x}$ for $x \geq 0$
5. `static double random()`: returns random $0.0 \leq r < 1.0$

Remark 19. For a random value in the range low $\leq x <$ high, use:

```
double x = (high - low) * Math.random() + lowValue;
```

For a random integer from 0 to $k - 1$, use:

```
int n = (int) (Math.random() * k)
```

5 Program Design and Analysis

See like literally the only table in these notes.

Vocab	Meaning
software development	writing a program
object-oriented programming	uses interacting objects
program specification	description of a task
program design	a written plan, an overview of the solution
program implementation	the code
test data	input to test the program
program mainteneance	keeping the program working and up to date
top-down development	implement main classes first, subsidiary classes later
independent class	doesn't use other classes of the program in its code
bottom-up development	implement lowest level, independent classes first
driver class	used to test other classes, contains `main` method
inheritance relationship	*is-a* relationship between classes
composition relationship	*has-a* relationship between classes
inheritance hierarchy	inheritance relationship shown in a tree-like diagram
UML diagram	Tree-like representation of relationship between classes
data structure	Java construct for sorting a data field (e.g. array)
data encapsulation	hiding data fields and methods in a class
setpwise refinement	breaking methods into smaller methods
procedural abstraction	using separate methods to encapsulate each task
algorithm	step-by-step process that solves a problem
stub method	dummy-method called by another method being tested
debugging	fixing errors
robust program	screens out bad input
compile-time error	usually a syntax error; prevents program from compiling
syntax error	bad language usage (e.g. missing brace)
run-time error	occurs during execution (e.g. `int` division by 0)
exception	run-time error thrown by Java method
logic error	program runs but does the wrong thing

Table 1. Vocab

6 Arrays and Array Lists

6.1 Arrays

Definition 25. *An array a data structure to implement a list of objects.*

Definition 26. *Subscripts are index values of an array, range from* 0 *to* $N-1$.

Example 9. Ways to initialize arrays:

1. `double[] data = new double[25];`
2. `double data[] = new double[25];`
3. `double[] data;`
   `data = new double[25];`
4. **initializer list**:
   `double[] data = 1.0, 5.0, 10.0, 25.0;`

Elements automatically initialized to

1. 0 for `int` and `double`
2. `false` for `boolean`
3. `null` for object references

Arrays have constant length, accessed through `arr.length`.

Arrays are objects, so references are passed. Thus, elements of the actual array can be accessed and modified.

Insertion and deletion of an element in an ordered list is inefficient.

6.2 ArrayLists

Definition 27. *An ArrayList is an alternative way to store a list of objects.*

ArrayLists have the following advantages:

1. shrinks and grows as needed
2. last filled slot is always `list.size() -1`, whereas must keep track for array
3. efficient insertion/deletion
4. easier to print ArrayList elements. Just do:
   `System.out.print(list);`

ArrayLists are part of the `java.util` package. Has the generic type `ArrayList<E>`, where E can be any type of object.

Methods:

1. `ArrayList()`: constructs empty list
2. `int size()`: returns number of elements

3. `boolean add(E obj)`: appends obj to end of list, always returns true. If obj not type E, run-time exception.
4. `void add(int index, E element)` inserts element at index. Elements from index and higher add 1 to indices.
5. `E get(int index)`: returns element at specified index.
6. `E set(int index, E element)`: replaces item at index with element. Returns the element previously at index. If element not type E, run-time exception.
7. `E remove(int index)`: removes and returns element and index. Elements to the right subtract 1 from indices.

Remark 20. Each method with an index parameter throws `IndexOutOfBoundsException` if index out of range. It is okay for `add()` if element is at the end of list.

ArrayLists can't use primitives, instead they used wrapper classes like `Integer` and `Double`. Automatic unboxing occurs when retrieving. If program tries to auto-unbox `null`, `NullPointerException` occurs.

Remark 21. Trying to add or delete element during enhanced for loop may result in `ConcurrentModificationException`.

6.3 Two-Dimensional Arrays

Example 10. Ways to declare 2-D arrays:

1. `int[][] table;`
2. `int[][] matrix = new double [3][4];`
   This makes the following array: $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
3. `int[][] mat ={{3, 4, 5}, {6, 7, 8}};`

Example 11. For the matrix `mat`: $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$
Then `mat` contains three arrays, ex: `mat[0]` contains {`1, 2, 3, 4`}.

For row/column lengths:

1. `mat.length` gives 3, number of rows.
2. `mat[1].length` gives 4, length of first row.

Definition 28. *A row-major traversal goes row by row, left to right. A column-major traversal goes column by column, top to bottom.*

7 Recursion

Definition 29. *A recursive method is a method that calls itself. It must have:*

1. *a base case/termination condition*
2. *a nonbase case whose actions move algorithm toward the base case*

Definition 30. *Tail recursion occurs in a method that has no pending statements following the recursive call.*

Remark 22. Infinite recursion results in running out of memory, `java.lang.StackOverflowError...`

Remark 23. Divide and conquer algorithms have the recursive call in front of pending statements, and every recursive algorithm can be written iteratively.

8 Sorting and Searching

8.1 Selection and Insertion Sorts

Definition 31. *Selection sort is an algorithm that:*

1. *finds smallest element in array and swaps with* `a[0]`
2. *finds next smallest and swaps with* `a[1]`*, and so on*

Remark 24. Notes:

1. for n elements, sorted after $n-1$ passes
2. after kth pass, first k elements are in final position.

Definition 32. *Insertion sort is an algorithm that:*

1. *inserts* `a[1]` *into position relative to* `a[0]`
2. *inserts* `a[2]` *into position relative to* `a[1]` *and* `a[0]`*, and so on.*

Remark 25. Notes:

1. for n elements, sorted after $n-1$ passes
2. after kth pass, `a[0]` through `a[k]` are sorted with respect to each other, but not necessarily in final positions
3. worst case: reverse order, leads to maximum comparison and moves
4. best case: already sorted, just one comparison, no moves.

8.2 Merge Sort and QuickSort

Definition 33. *Merge sort works in the following manner:*

1. *break array into halves (if more than 1 element)*
2. *merge sort each half*
3. *merge two subarrays into sorted array*

Remark 26. Notes:

1. needs temporary array
2. not affected by initial ordering. best/worst/average cases similar

Definition 34. *QuickSort sorts an array as follows:*

1. *partition the array (if at least two elements)*
2. *quicksort left and right*

The partition function works as follows:

1. Choose a pivot
2. initialize `down` and `up` to have values 0 and $n-1$
3. increase `down` until `a[down]` $>$ pivot, and decrease `up` until `a[up]` $<$ pivot
4. swap `a[down]` and `a[up]`
5. continue 3-4 until `down` $=$ `up`

Remark 27. Notes:

1. for fastest runtime, partition into two roughly same size
2. worst case if repeatedly divides into 1, $n-1$. some avoid this by choosing pivot with more information.

8.3 Searching Algorithms

Definition 35. *Sequential search searches through a list by comparing each element to the key, in order.*

Best case 1, worst case n, average $n/2$ comparisons.

Definition 36. *Binary search is a divide-and-conquer approach that*

1. *compares key to middle value of list*
2. *chooses only the half of the array containing the key, then repeats*

Remark 28. Note:

1. Best case 1, worse case $\lfloor \log_2(n) + 1 \rfloor$
2. worst case is always endpoint on right end, or not in array
3. any key $< a[0]$ or $> a[n-1]$ is worst case, but if within may or may not be worst case