JAVA

Lecture X – Generic

ARRAYLIST

- What is the problem of array?
- Cannot change its size
- We are asked to pick up all the numbers that is greater than 10, but we don’t know how many integers are there.
- Use java. util.ArrayList, can be seen as dynamic array
- ArrayList provides a wide range of useful methods to manipulate a collection of elements.

```
ArrayList<Type> name = new ArrayList<>();
```

ARRAYLIST

- Useful ArrayList methods:

```
boolean add(E e); // add an element to the tail
void add(int index, E e); // add an element at a specified position
void clear();// remove all elements
boolean contains(Object o); // check if it contains a specified element
E get(index i);// get an element at a specified position
E remove(index i);// remove an element at a specified position
boolean remove(Object o);// remove the first occurrence of an object
int size();// return the size of the arraylist
```

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ArrayList--

EXAMPLE: ARRAYLIST

```
class ArrayListDemo{
      public static void main(String[] args){
            ArrayList<Character> a1 = new ArrayList<>();
            a1.add(‘A’);
            a1.add(‘B’);
            a1.add(1, ‘C’);
            a1.remove(2);
            …
      }
}
```

TYPE CONVERSION

- Can we read integer and floating numbers from the command-line?

```
java Args 1 2
```

- We want 1 and 2 as integer number!

- Automatic type conversion:
  - Two types are compatible.
  - The destination type is larger than the source type.
  - E.g., byte to int, int to long, long to double, …
- Cast: an instruction to the compiler to convert one type into another

```
(target-type) expression
```

TYPE CONVERSION

```
double x, y;
// …
int z = (int) (x / y);
```

- Narrowing conversion: information might be lost.
- E.g., information lost when we convert long to short

- Example: CastDemo

- How to convert string into integer, double, …

TYPE WRAPPER

- Type Wrapper: classes that encapsulate the primitive types.
- Primitive types: are not objects, e.g., cannot be passed by reference.
- Double, Integer, Float, …

- Numeric wrappers provide methods to convert a string into corresponding number.
- Double.parseDouble(String)
- Integer.parseInt(String)
- Short.parseShort(String)
- …
- Boolean values:
- Boolean.parseBoolean(String)

ARRAYLIST

- Useful ArrayList methods:

```
boolean add(E e); // add an element to the tail
void add(int index, E e); // add an element at a specified position
void clear();// remove all elements
boolean contains(Object o); // check if it contains a specified element
E get(index i);// get an element at a specified position
E remove(index i);// remove an element at a specified position
boolean remove(Object o);// remove the first occurrence of an object
int size();// return the size of the arraylist
```

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ArrayList--

GENERICS

- Which methods are different from what we haven seen before?

GENERICS

- Which methods are different from what we haven seen before?
- `boolean add(E e) // what is type E?`
- `E get(int index) // what is returned?`
- Are ArrayList<Integer> and ArrayList<Character> the same?

GENERICS

- Which methods are different from what we haven seen before?
- `boolean add(E e) // what is type E?`
- `E get(int index) // what is returned?`
- Are ArrayList<Integer> and ArrayList<Character> the same?
- NO, they are different types.
- Generics: the capability to parameterize types.
  - E is a generic type, i.e., e is an object of the generic type
  - Automatically work with the type of data passed to its type parameter.

EXAMPLE: GENERICS

```
class Gen<T>{ // T is a generic type
      T ob; // declare a reference to an object of type T
      Gen(T o){ // constructor
            ob = o;
      }
      T getOb(){
            return ob;
      }
      void showType(){
            System.out.println(ob.getClass().getName());
      }
}
```

OBJECT VS GENERICS

- Object is the ancestor of all classes, i.e., all classes can be seen as an object .
- Can we use Object to generalise classes, interface and methods?
- `Object method(Object o)`

OBJECT VS GENERICS

- Object is the ancestor of all classes, i.e., all classes can be seen as an object .
- Can we use Object to generalise classes, interface and methods?
- `Object method(Object o)`

- What are the problems?
- Object cannot be used to safely convert to its actual type.
- e.g., `Object method(Object o)` always returns an Object but not its actual type.
- Unless o is casted to its actual type, it cannot call its own methods.

GENERICS

- General form:

```
class class-name<type-param-list>{
     body…
}
```

- Properties on Generics:
  - Works only with Object type, e.g., why ArrayList<int> fails?
  - Different versions of a generic type are not type-compatible, e.g., ArryList<Integer> and ArrayList<Double>
  - We can use more than one generic types, e.g,. Gen<V, T>

EXAMPLE

```
class NumType<T>{
      T num;
      NumType(T t){
            num = t;
      }
      multiply double(double x){
            return num.doubleValue() * x;
      }
}
```

BOUNDED TYPES

```
class NumType<T extends Number>{
      T num;
      NumType(T t){
            num = t;
      }
      double multiply(double x){
            return num.doubleValue() * x;
      }
}
```

BOUNDED TYPES

```
      class NumType<T extends Number>{
            T num;
            NumType(T t){
                  num = t;
            }
            boolean absEquals(NumType<T> ob){
                  return Math.abs(num.doubleValue()) ==
Math.abs(ob.num.doubleValue());
            }
      }
```

What is the problem?

WILDCARD ARGUMENTS

```
      class NumType<T extends Number>{
            T num;
            NumType(T t){
                  num = t;
            }
            boolean absEquals(NumType<?> ob){
                  return Math.abs(num.doubleValue()) ==
Math.abs(ob.num.doubleValue());
            }
      }
```

What is the problem?

GENERIC METHODS

It is possible to have generic method defined in a non-generic class

```
class GenericMethod{
      static <T, V> boolean arrayEquals(T[] x, V[] y){
            …
      }
}
```

GENERIC CONSTRUCTOR

Also it is possible to have generic constructor

```
class Summation{
        private int sum;
        <T extends Number> Summation(T arg){
                for(int i = 0; i < arg.intValue(); i++){
                        sum += i;
                }
        }
}
```

STRING

- One of the most important data structure in Java.
  - Strings are objects in Java, not primitive type.

- Construct a String:
- `String str = new String("Happy"); // like an object`
- `String str2 = new String(str); // from another string`

- Alternatively
- `String str = "Happy";`

STRING METHODS

- Useful methods that operate on String:
- `boolean equals(Object str)//` return true, if they contains the same character sequence
- `int length() //` return the number of characters
- `char charAt(int index) //` return character at a specified index
- `int compareTo(String str) //` comparison based on Unicode of each character
- `int indexOf(char ch/ string str) //` return the index of the first occurrence of the given character or substring
- `int lastIndexOf(char ch/ string str) //` last index of..

- You can find more here
- https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

IMMUTABLE STRING

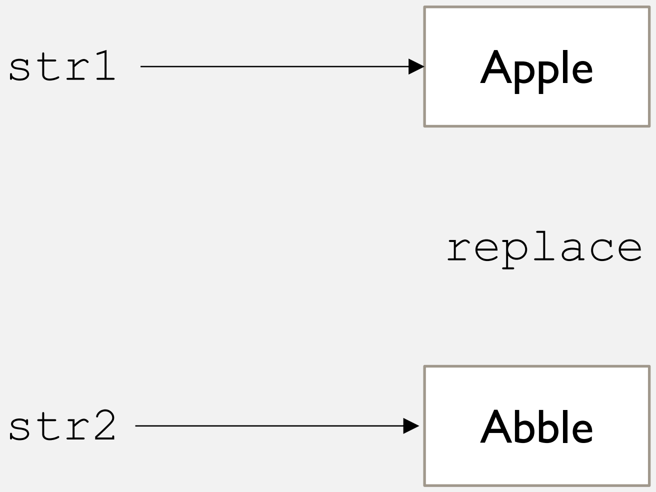
- The contents of a String object are immutable.
- Example, replace returns a string resulting from replacing all occurences of oldChar in this string with newChar

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

String str1 = “Apple”;

str1.replace(’p’, ‘b’); // will it change the value of str1?

String str2 = str1.replace(’p’, ‘b’);
```

IMMUTABLE STRING

- The contents of a String object are immutable.
- Example, replace returns a string resulting from replacing all occurences of oldChar in this string with newChar

```
String replace(char oldChar, char newChar)
String str1 = “Apple”;
str1.replace(’p’, ‘b’);
String str2 = str1.replace(’p’, ‘b’);
```



Once String “Apple” is created, it cannot be changed.

```
str1 = str1.replace(’p’, ‘b’);
// update the reference, “Apple” Garbage Collected
```

What if we need to manipulate a string in several steps? StringBuffer, StringBuilder

STRING AND CHAR[]

- Both String and char[] represent a collection of characters, are they the same?

```
String str1 = “Happy”;
for(char c : str1){
      ...
} // Will it work?

char[] cs1 = {‘a’, ‘b’, ‘c’};
char[] cs2 = {‘d’, ‘e’, ‘f’};
char[] cs3 = cs1 + cs2; // Will it work?
```

STRING VS CHAR[]

- Other differences:

- Data type?
- Immutable?
- Build-in functions?
- Accessing each character?
- Conversions?

STRING VS CHAR[]

- Data type: Single data type vs collections
- Immutable: Immutable vs mutable
- Build-in functions: String has a lot of build-in functions, char[] not.
- Accessing each character: charAt() vs var_name[index]
- Conversions:

```
String s = “Happy”;
char[] cs = s.toCharArray();
cs = {‘H’, ‘A’, ‘P’, ‘P’, ‘Y’};
s = new String(cs);
```

COURSEWORK 1

- Step 1: Read input from a particular file (specified by its location).
- Step 2: Translate each line into mathematic formula.
- Step 3: Write the results into a file called “Out.txt”.

- Note, all exceptions need to be handled. Required error message needs to be shown for certain exceptions.

PSEUDO CODE

```
procedure calculate(path)
    lines <- readFromFile(path)
    results <- Ø
    for each line in lines do
        result <- evaluate(line)
        results.add(result)
    endfor
    writeToFile(results, ”Out.txt”)
```

JAVA CODE

```
public void calculate(path){
      ArrayList<String> lines = readFromFile(path)
      ArrayList<String> results = new ArrayList<String>();
      for(int i = 0; i < lines.size(); i++){
            results.add(evaluate(lines.get(i)));
      }
      writeToFile(results, “out.txt”);
}
```

SIMPLE INPUT AND OUTPUT

```
private ArrayList<String> readFromFile(String path){

        …

}


private writeToFile(ArrayList<String> results, String path){

        …

}
```

EVALUATE

- 1. Check if the given string satisfy the syntax of Arithmetic Expression.
- 2. Extract and convert strings into Mathematical Operators.
- 3. Extract and convert strings into integers. (Type conversion)
- 4. Calculate the results. (Need to consider the precedence)

- Note, we need to think about how to check the validation of the expression and how to store and sort the operators and numbers.

EVALUATION

- Alternatively
- Syntax of the formula
  - “Num1 Ops Num2 Ops Num3 … Ops Numn”
  - “Num1”
- More precisely, if we use DM to represent the results of multiply and division
  - MD :== Num1 *|/ Num2 *|/ Num3 … *|/ Numn
- Then a formula is defined as:
  - Formula :== MD1 +|- MD2 +|- MD3 … +|- MDn

EVALUATION

- Alternatively
- Syntax of the formula
  - “Num1 Ops Num2 Ops Num3 … Ops Numn”
  - “Num1”
- More precisely, if we use DM to represent the results of multiply and division
  - MD :== Num1 *|/ Num2 *|/ Num3 … *|/ Numn
- Then a formula is defined as:
  - Formula :== MD1 +|- MD2 +|- MD3 … +|- MDn

EVALUATION

- Does order matter?
  - + is commutative and associative, so order does not matter.
  - - is commutative but not associative, so order matters.
  - * is commutative and associative, however the order still matters.
  - / is commutative but not associative, so order matters.
- Solution: (first, we need to get ride of all these spaces)
  - Step 1: split based on +. Each member is either MD or MD-MD…
  - Step 2: split based on 0. e.g., {[MD,MD], [MD], [MD, MD, MD], …}   Similar to CNF
  - Step 3: evaluate MD

PSEUDO CODE

```
procedure evaluate(s)
    str <- removeSpace(s)
    as <- split(str, “+”)
    result <- 0
    for each a in as do
        ms <- split(a, “-”)
        result <- result + ms[0]
        for i <- 1, ms.length-1 do
            result <- result – evaluateMD(i)
        endfor
    endfor
    return result
```

PSEUDO CODE

```
procedure evaluateMD(str)
    ms <- split(str, “*”)
    result <- 1
    for each m in ms do
        ds <- split(m, “/”)
        result <- result * ds[0]
        for i <- 1, ds.length-1 do
            num <- toInt(ds[i])
            result <- result / num
        endfor
    endfor
    return result
```