CSE 11
Accelerated Intro to Programming
Discussion Section 8

Shihua Lu, Spring 2021

Logistics

- PA7 due today at 11:59PM
- PA8 released

Generic Methods

- Enable us to run the same code for multiple data types
- we use <> to specify parameter types in generic method and class creation
- The type parameter's name is our choice, such as T, U

Example : generic method to print elements in array

```
public class GenericMethodTest {
  public static < E > void printArray( E[] inputArray ) {
    for(E element : inputArray) {
      System.out.println(element);
    }
  }
  public static void main(String args[]) {
    // Create arrays of Integer, Double and Character
    Integer[] intArray = { 1, 2, 3, 4, 5 };
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
    Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
    printArray(intArray);  // pass an Integer array
    printArray(doubleArray);  // pass a Double array
    printArray(charArray);  // pass a Character array
  }
}
```

return type

placeholder for the actual type

T
A

char

primitive type X

(int)[] intArray = {1,2,3};
printArray (int Array);

X

Boolean [] boolArray = { true, false, true } ?

Generic Interfaces and Classes

- write it in front of the name of the interface or class instead

```
interface Checker<T> {
    boolean check(T t); // We can use T here.

}
```

- notable difference between generic methods and classes is that when using generic classes/interfaces, Java won't figure out the actual types for us, and we need to provide them manually

```
class LongTweet implements Checker<TextTweet> { // We need to provide the TextTweet type here ...
    public boolean check(TextTweet t) {

        return t.content.length() > 20;

    }

}
```

Lists and ArrayLists

(handwritten: interface)

(handwritten: class that implements List)

- ArrayList is an implementation of List interface provided by Java
- we used regular arrays to store lists of data, read them, and modify them. ArrayLists are a generic type in Java that allows us to do all the same things, with slightly different syntax:

(handwritten: check documentation)

(handwritten: arrays) *(handwritten: lists)* *(handwritten: built-in)* *(handwritten: Strings, doubles, booleans)*

int[] numArray = {1, 2, 3};	List<Integer> numList = Arrays.asList(1, 2, 3);
int one = numArray[0];	int anotherOne = numList.get(0);
numArray[1] = one * 2;	numList.add(1, anotherOne * 2);
for (int num: numArray) { /* ... */}	for (Integer num: numList) { /* ... */}

(handwritten: index) *(handwritten: numList.get(1);)* *(handwritten: value)*

(handwritten: int) *(handwritten: Integer)*

Lists and ArrayLists

- The key distinction between arrays and ArrayLists is that ArrayLists are resizable, so we can add and remove values from them

```
List<String> messages = Arrays.asList("Hello", "CSE11");

// Appends "on your assignments!" to the end of messages
messages.add("on your assignments!");

// Inserts "Good luck" at index 2, shifting "on your assignments!" to the right.
messages.add(2, "Good luck");

// Removes "Hello" from the list, and returns it.
String hello = messages.remove(0);
```

Check documentation

Overloading — PA 8

- providing multiple implementations of the same method, differentiated by the parameters they accept

  int add(int a, int b) { /* ... */}

  double add(double a, double b) { /* ... */}

- only works if the different method have a different number of parameters, or parameters of different types , but not if they only have different return types ✗
- A common use for overloading methods is to provide default parameters. For instance, the following method is overloaded to use a default length of 5

  /*

     This method returns a new ArrayList containing the Strings in `strs` that

     are longer than `length`.

  */

  static ArrayList<String> longStrings(ArrayList<String> strs, int length) {   2 parameters

  ...

  }

  static ArrayList<String> longStrings(ArrayList<String> strs) {   1 parameter

     return longStrings(strs, 5);

  }

     default argument to

Exceptions

compile error

javac Program.java

run-time *java Program*

- Java's way of handling errors that can happen when we run our programs
- run into exceptions as a result of incorrect/buggy code as well. Some of these exceptions include: *Check documentation*
  - ArrayOutOfBoundsException: Happens when we try to get the element at an index of an array that doesn't exist
  - NullPointerException: Happens when we try to read a field of, or call a method on a null value.
  - ArithmeticException: Happens when we try to run an invalid arithmetic operation (e.g. divide by zero) *IOException*
- we can provide a custom, more descriptive, error message while throwing exceptions

```
static Integer max(ArrayList<Integer> elements) {
    if (elements.size() == 0) {
        throw new IllegalArgumentException("max got an empty list");
    }

    /* ... */
}
```

keyword

Error message

throw SomeException;

try {

/ operation */*

} catch (Exception e) {

/ handle exception */*

}

PA8

- Start early!

Thanks!