Midterm

1. In Java, as in most programming languages, * is multiplication, + is addition, - is subtraction, and / is division. What does that % operator do?

The remainder operator % divides number1 by number2 (rounds floating-point numbers to integers), and then returns only the remainder as result. The remainder operation in Java has the following properties: for all int values A and all non-zero int values B: $(A/B) * B + (A\%B) == A$. This means that when the remainder operation returns a nonzero result, it has the same sign as the left operand.

2. What is the difference between an overridden and overloaded method?

Overloaded:

(1) Method overloading is a means for classes to handle different types of data in a unified manner. Multiple functions of the same name exist at the same time, with different number / type of parameters. (Overloading is an expression of polymorphism in a class.)

(2) Java method overloading means that multiple methods can be created in the class, they have the same name, but have different parameters and different definitions. When calling a method, the number of different parameters and parameter types passed to them is used to decide which method to use. This is polymorphism.

(3) When overloading, the method name must be the same, but the parameter type and number are different, and the return value type can be the same or different. The return type cannot be used as a criterion for distinguishing overloaded functions.

Overridden:

(1) The polymorphism between the parent class and the child class redefines the parent class function. If a method defined in a subclass has the same name and parameters as its parent class, we say that the method is overridden. In Java, subclasses can inherit the methods in the parent class without rewriting the same method. But sometimes subclasses do not want to inherit the methods of the parent class intact, but want to make certain modifications, which requires the use of method rewriting.

(2) If the method in the subclass has the same method name, return

type, and parameter list as a method in the parent class, the new method will overwrite the original method. If you need the original method in the parent class, you can use the super keyword, which refers to the parent class of the current class.

(3) The access modification permission of the subclass function cannot be less than that of the parent class;

In summary, the same method can make different treatments according to the different input data, that is, the method overload has different parameter lists (static polymorphism).

And when the subclass inherits the same method from the parent class, the input data is the same, but to make a response that is different from the parent class, you have to override the parent class method, that is, override the method in the subclass. Same parameters, different implementations (dynamic polymorphism).

3. When multiplying an int value and a float value, what will be the type of the result?

The result will be a float type. Transition to the highest precision among the types of operands participating in the operation, so the answer is the float type. If there is a decimal to participate in the operation, it is of type double, because java default decimal direct quantity is of type double.

4. You have an array of objects called "objs" that contains a list of Integer objects. It was declared like this: $Object[]\ \ objs$ **It doesn't matter how the Integer objects got there or what their values are. You have a variable int intVal. Write some code (this should just take at most 2 lines, but you could do it in 1 line) that gets the integer value of the first object in the objs array and assigns that value to intVal. .**

$int\ \ intVal\ \ =\ \ (int)\ \ objs[0];$

5.Take that object from the array.The result of (new Object()).toString() is something like java.lang.Object@15db9742, let's say we execute the following:

$$objs[5] = newInteger(20); ObjectmyObj = objs[5];$$

What is the output of myObj.toString()? Explain why.

The output is 20. Because Integer is a subclass of Object, and toString () function is overridden.

The toString method of class Object returns a string that includes the name of the class for which the object is an instance, the symbol "@" and the unsigned hexadecimal representation of the object's hash

"java.lang.Object@15db9742" gets from "code.getClass().getName() + '@' + Integer.toHexString(hashCode())"

6.You have two strings: String s1 = "Alice" and String s2 = "Bob" What is the result of (s1 == s2) ? What is the result of s1.compareTo(s2) ? Explain your answers – why each gives the result it does and what that means.

The answers are false and -1 . "==" compares the referenced address in memory in java. s1 and s2 have different addresses, so the answer is false. The compareTo () method compares two strings in lexicographic order. The s1 string is less than the s2 string, so it returns a value less than 0;

7. Remember in an earlier lecture when discussing Abstracts, Interfaces, and Generics we had the class Fruit, declared as: public abstract class Fruit public Fruit(String name) /*etc*/ /* etc */ . We created Apple and Orange objects. Why didn't we ever simply do the following: Fruit f = new Fruit("Apple"); ?

Fruit is an abstract class and an abstract class cannot be instantiated. By the way, we need the Apple class and the Orange class to have different operations, not just different in name, so it is inconvenient to simply use a Fruit class to complete. But they also have some common attributes, so we abstracted a fruit class, and wrote apple and orange classes as its subclasses.

8. In an earlier lecture, we had two abstract classes, Animal and Fruit, from which we made various subclasses. Some of those subclasses implemented the interface Edible. Why did we choose to make classes subclasses of Animal and Fruit, but have Edible as an interface? To put the question more generally, why would you design an object as a subclass of another object vs. why would you create an interface and have an object implement an interface? (There's no single right answer to this question, but there are definitely wrong answers)

The methods in the interface must be abstract, and there are only static variables and final variables in the interface. But the weight of the animal (the name of the fruit) cannot be static and final, because different animals have different weights. Moreover, there is no construction method in the interface. We can implement the same method in the super class, such as setweight () in the Animal class, if Animal is an interface, it can not be so. So, we design an object as a subclass of another object.

But HowToEat can be one of the common methods of fruit class, so we can use the interface and implement it in the fruit subclass.

9. Assume there is a file data.txt that should contain two integers per line (eg, a sample line is "5 10"). This program reads in each line and divides the first number by the second number.

a) This is not going to compile because of the IOExceptions raised by the FileReader and BufferedReader constructors. Modify the code so it will compile.

```
Class Main {
   public static void main(String[] args) {
      try {
         file = new FileReader("data.txt");
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      }
      BufferedReader fileInput = new BufferedReader(file);
      String inLine = null;
      try {
         inLine = fileInput.readLine();
      } catch (IOException e) {
         e.printStackTrace();
```

```
        }
        while (inLine!= null) {
           String[] numbers = inLine.split(" ");
           Integer a = Integer.parseInt(numbers[0]);
           Integer b = Integer.parseInt(numbers[1]);
           Integer c = a/b;
           System.out.println("result = " + c);
           try {
              inLine = fileInput.readLine();
           } catch (IOException e) {
              e.printStackTrace();
           }
        }
     }
  }
```

b) Add a try/catch block in the while loop to catch any Runtime exceptions. What Runtime exceptions could be raised that you might want to look out for?

(1)java.lang.NumberFormatException

if inLine has no Integer, then "Integer a = Integer.parseInt(numbers[0]);" will have NumberFormatException.

(2)java.lang.ArrayIndexOutOfBoundsException

if numbers has only one element, then "Integer b = Integer.parseInt(numbers[1]);" will have ArrayIndexOutOfBoundsException.

(3)java.lang.ArithmeticException

If b == 0, then "Integer c = a / b;" will have the problem of "ArithmeticException: / zero".

10. If we see a method and see which arguments it takes, how do we know if we can pass in a Lambda function? Specifically, what kind of parameter type does a method have to accept for you to pass in a lambda function?

The parameter type is functional interface. For the compiler to understand lambda expressions, the interface must contain exactly one abstract method. If it contains multiple methods, the compiler will not be able to compile the lambda expression. Such an interface is known as a functional interface.

11. Look at this code. The msg field of TimerMessage is private. In the main method of the TimerMessageRunner class, if I executed System.out.println(tm.msg), I would get an error. However, the TimePrinter method can access msg from the TimerMessage class without any problem. Why?

All member variables modified by private are called private variables. It only allows access within this class, no external class can access it. The TimePrinter method is a member function of the TimerMessage class, so it can access the private variable msg of the TimerMessage class. But outside the TimerMessage class, the private variable msg is not directly accessible to the TimerMessageRunner class.

12. The following code opens a window with a "Close" button, but it does nothing. What do you need to add to the code so that when you press the "Close" button, it executes System.exit(0), terminating the application?

```
   public TestFrame(){
      closeButton.addActionListener((e) -> System.exit(0) );
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      this.setLayout(new FlowLayout());
      this.add(closeButton);
}
```

add code "$closeButton.addActionListener((e)->System.exit(0))$;" in the public method TestFrame()