Homework_02

1. **What is Interface and what is abstract class? What are the differences between them?**

   Interface: an abstract type which contains a collection of methods and constant variables.

   Abstract class: a restricted class that cannot be used to created objects (to access it, must be inherited from other class).

   Differences between them:

	Interface	**Abstract class**
Method	abstract	abstract and non-abstract
Variables	static and final	final, non-final, static and non-static
Access	members of interface are public by default	can have class members like private, protected, etc
Implementation	can extend another interface only	can extend another Java class and implement multiple Java interfaces

2. **What is the differences between overriding and overloading?**

	Overriding	**Overloading**
Definition	methods have same signatures (name and parameters) in the superclass and the child class	methods have same nabe but different parameters in the same class
Polymorphism	runtime	compile time
Access	members of interface are public by default	can have class members like private, protected, etc
Break	cause serious issues in the program because the effect will be visible at runtime	generate compile-time error and easy to fix

3. **What is final key word? (Filed, Method, Class)**

Final	**Contexts**
Field/variables	create constant variables
Method	prevent method overriding
Class	prevent inheritance

4. **What is Java garbage collection?**

   Java garbage collection is the process by which Java programs perform automatic memory management.Java programs compile to bytecode that can be run on a Java Virtual Machine(JVM). When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

5. **What is the differences between super and this?**

"Super" keyword is used to access methods of the parent class while "this" is used to access methods of the current class.

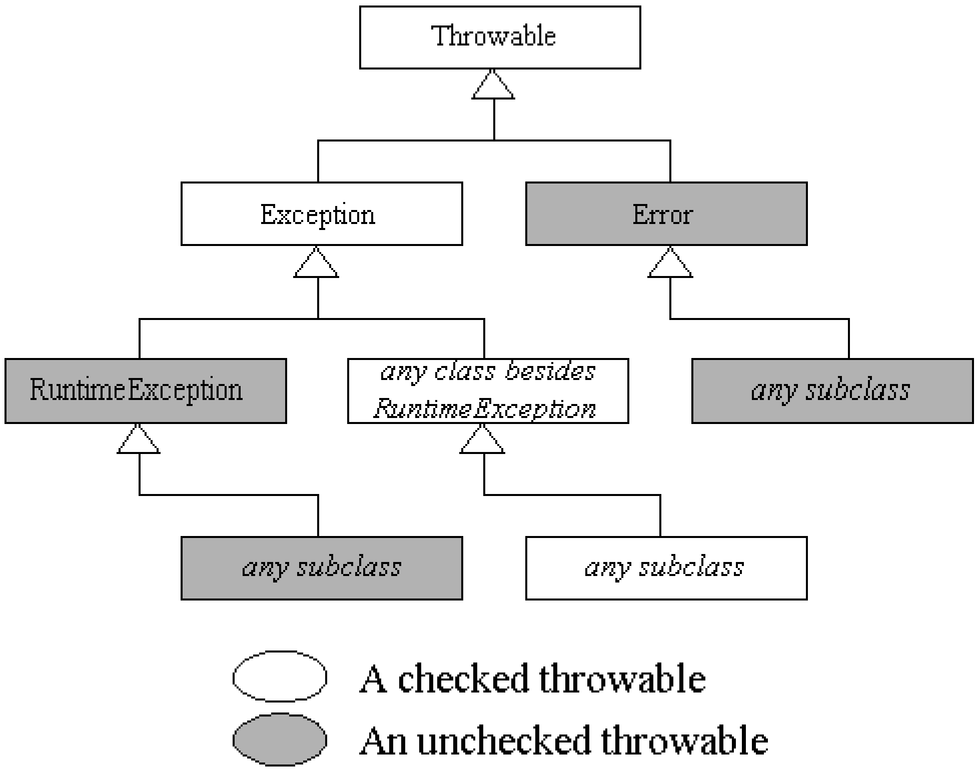
6. **Can we use this keyword in constructor and why?**

   Yes, "this" can be used within a constructor to call another constructor in the same class

7. **What is Runtime/unchecked exception? what is Compile/Checked Exception?**

   Unchecked exception: Error + RuntimeException. It occurs at the time of execution.These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation. The compiler doesn't require methods to catch or to specify (with a throws) them.

   Checked exception: All the Exception but RuntimeException. It occurs at the compile time. These exceptions cannot simply be ignored at the time of compilation.**



8. **what is the difference between throw and throws?**

   The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

   Throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

9. **Run the below three pieces codes, Noticed the printed exceptions. why do we put the Null/Runtime exception before Exception?**

   If parent exception (Here, Exception) is put before its child class of exception (here, Null/Runtime exception), the later exception will not be caught.

```
public class Main {
    public static void main(String[] args) {
        int a = 0;
        int b = 3
        String s = null;
```

```java
        try {
            System.out.println(b / a);
            System.out.println(s.equals("aa"));
            throw new RuntimeException();
        } catch (ArithmeticException e) {
            e.printStackTrace();
        } catch (NullPointerException e) {
            e.printStackTrace();
        } catch (RuntimeException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.getMessage();
        }

        System.out.println("End ...");
    }
}
public class Main {
    public static void main(String[] args) {
        int a = 0;
        int b = 3
        String s = null;
        try {
            // System.out.println(b / a);
            System.out.println(s.equals("aa"));
            throw new RuntimeException();
        } catch (ArithmeticException e) {
            e.printStackTrace();
        } catch (NullPointerException e) {
            e.printStackTrace();
        } catch (RuntimeException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.getMessage();
        }

        System.out.println("End ...");
    }
}
public class Main {
    public static void main(String[] args) {
        int a = 0;
        int b = 3
        String s = null;
        try {
            // System.out.println(b / a);
            // System.out.println(s.equals("aa"));
            throw new RuntimeException();
        } catch (ArithmeticException e) {
            e.printStackTrace();
        } catch (NullPointerException e) {
            e.printStackTrace();
        } catch (RuntimeException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.getMessage();
        }

        System.out.println("End ...");
    }
}
```