Section 1 The "Googling" Part (20 points)

True/False: If the statement(s) is(are) true, circle T. Otherwise, circle F.

1. **T** **F** You can only construct a wrapper object from a primitive data type value. You cannot construct using a string representing the numeric value.

2. **T** **F** In Java String is a class.

3. **T** **F** Overriding is modifying the definition of a superclass method. Overloading is using the same name of a method but different parameter types.

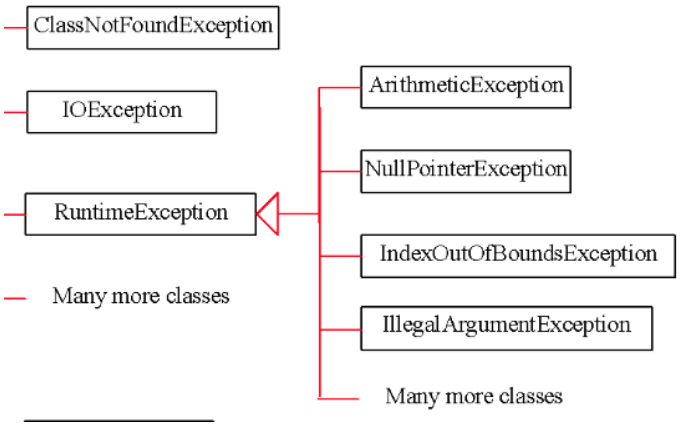
4. **T** **F** String s1 = "CSE 102 Midterm";
String s2 = "CSE 102 Midterm";
String s3 = new String("CSE 102 Midterm");
s1 == s3?

5. **T** **F** Every class in Java has a superclass. If no superclass is defined, Object is the superclass.

6. **T** **F** Like properties and methods, a superclass's constructors are inherited in the subclass.

7. **T** **F** Explicit casting must be used when casting an object from a superclass to a subclass.

Use the given UML class diagram to the right to answer questions 8-10



8. **T** **F** RuntimeException instanceof NullPointerException

9. **T** **F** NullPointerException instanceof ArithmeticException

10. **T** **F** ArithmeticException instanceof RuntimeException

Multiple Choice: Circle the answer that is **most correct**.

11. The keyword *super* can be used to do which of the following:
    a. To override a superclass method
    b. To overload a superclass method
    c. To convert an object to an object of the superclass type
    d. To call a superclass method
    e. All of the above
12. Which of the following about the replace method is **true**:
    a. It is a method in the String class.
    b. It changes the String object to a new value
    c. It is a void type (it returns nothing)
    d. If the character to be replaced is not found, it throws an Exception
    e. All of the above
13. In order to use the java.util.Arrays.sort(array) method to sort an array, the elements in the array must be instances of which of the following:
    a. Sortable<E>
    b. Arrays<E>
    c. Integer<E>
    d. Number<E>
    e. Comparable<E>
14. A String can be constructed in all of the following ways **except**:
    a. String a = String("Choose me");
    b. String b = new String("No, choose me");
    c. String c = "I am correct";
    d. String d = new String();
    e. All of these can be used
15. Which of the following statements is **not true**
    a. A non-abstract class can have an abstract superclass
    b. A non-abstract class can have an non-abstract superclass
    c. An abstract class can have an abstract superclass
    d. An abstract class can have a non-abstract superclass
    e. All are true

Matching: Write the letter of the Java keyword on the right in the blank next to the definition for its **most common use**.

E	16. A block of code in which there could be an Exception	A. catch
A	17. A block of code to process an Exception (if one happens)	B. finally
B	18. A block which will execute no matter what (even if an Exception occurs)	C. implements
D	19. Indicates this is a subclass of a superclass	D. extends
C	20. Indicates this class uses the methods/properties of an interface	E. try

Section 2 Can We Divide By Zero for Very Large Values of Zero? (20 points)

In the box provided, write the output generated by the given code segment.

```
public static void main(String[] args){
   try{
      myMethod(1, 4);
      myMethod(5, 2);
      myMethod(9, 0);
   } catch (Exception ex){
      System.out.println("Error: In main method.");
   }
}

public static void myMethod(int a, int b) throws ArithmeticException {
   try {
      if(b == 0)
         throw new ArithmeticException();
      System.out.print(a / b);
      System.out.println(" R: " + (a % b));
   } catch (ArithmeticException ex){
      System.out.println("Error: b cannot be 0.");
   } finally {
      System.out.println("finally");
   }
}
```

Output:

```
0 R: 1
finally
2 R: 1
finally
Error: b cannot be 0.
finally
```

Section 3 If I Call Myself, Will I Answer? (25 points)

In the box provided, write a **recursive** method called *repeatChar(char c, int n)* that returns a String that is the character *c* repeated *n* times.

```
public static String repeatChar(char c, int n){

}
```

In the box provided, write a **recursive** method called *repeatString(String s, int ns)* that takes the individual characters of s and the digits of ns and repeats each character in *s* the respective digit in *ns* times. Examples below for both *repeatChar()* and *repeatString()*. You may use *repeatChar()* in your solution.

```
public static String repeatString(String s, int ns){

}
```

```
Midterm2018.repeatChar('A', 3)
AAA
Midterm2018.repeatChar('b', 6)
bbbbbb
Midterm2018.repeatString("CSE", 123)
CSSEEE
Midterm2018.repeatString("CSE", 102)
CEE
Midterm2018.repeatString("SE", 102)
EE
Midterm2018.repeatString("CSE", 12)
SEE
```

Section 4 OMG It's UML (35 points)

Draw a UML class diagram and write the java classes for the following system (All classes should be in a single file called Bank.java):

1. Classes
    a. A Bank has Staff and Accounts
    b. Staff can be Manager, Teller, or Security or they can just be a Staff
    c. Account must be either a Debit or Investment (Hint: we can not create an Account object)
2. Properties
    a. Staff has a name, age, and salary. Age and salary must both be positive.
    b. Manager has a count of employees under them. This number must be non-negative.
    c. An Account has an account number and an amount. The amount in the account must be non-negative.
    d. Investment has a decimal interest rate property. Interest rate must be between 0 exclusive and 1 inclusive (i.e. (0, 1]). If the interest rate is 5%, this value will be stored as 0.05.
3. Operations
    a. Staff has a *getRaise()* method that takes an integer parameter. This integer must be between 1 and 100. and represents the percentage increase in their salary. It should set the salary to the calculated amount.
    b. Manager has an *hireStaff()* method that increments the count of employees by one and a *fireStaff()* method that decrements the count of employees by one.
    c. Account has a *withdraw()* method that takes a decimal parameter. If there are enough funds in the account, it decreases the amount. If there are not enough funds, it throws an Exception.
    d. Account has a *deposit()* method that takes a decimal parameter. It adds to the funds in the account. If the amount passed is negative, an Exception should be thrown.
    e. Investment has an *accrueInterest()* method that increases the amount using the interest rate.

```
public class Bank {
     private Staff[] staffList;
     private Account[] acctList;

     public Bank() {
          staffList = new Staff[100];
          acctList = new Account[100];
     }
// Will need methods to add staff and accounts
}
```

```
class Staff {
	private String name;
	private int age;
	private double salary;

	public void getRaise(int raise) {
		salary = salary + salary * raise / 100;
	}
}
```

```
class Manager extends Staff {
	private int countEmp;

	public void hireStaff() {
		countEmp++;
	}

	public void fireStaff() {
		countEmp--;
	}
}
```

```
class Teller extends Staff {
}
```

```
class Security extends Staff {
}
```

```
abstract class Account {
	private String acctNum;
	private double amount;

	public Account(String acctNum) {
		this.acctNum = acctNum;
		this.amount = 0;
	}

	public void withdraw(double amount) {
		if (amount > this.amount)
			throw new IllegalArgumentException();
		this.amount -= amount;
	}

	public void deposit(double amount) {
		if (amount < 0)
			throw new IllegalArgumentException();
		this.amount += amount;
	}
}
```

```
class Debit extends Account {
}
```

```
class Investment extends Account {
      private double rate;

      public Account(String acctNum, double rate) {
            super(acctNum);
            this.rate = rate;
      }

      public void accrueInterest() {
            deposit(getAmount() * rate);
      }
}
```