GEREKSİNİMLER

Basit bir bankacılık uygulamasının, kullanıcı tanımlı aykırı durum denetimi (exception handling) sınıfları ile birlikte Java dili kullanılarak gerçekleştirimi istenmektedir. Gerçekleştirimi istenen basit uygulamaya ilişkin gereksinimler aşağıda verilmiştir:

* İlk olarak, banka hesapları üzerinde yapılan işlemler ile ilgili tüm aykırı durumların üst sınıfı durumunda olacak **"AccountException"** isimli sınıfı yazınız. Daha sonra bu sınıftan kalıtım yolu ile **"NegativeAmountException"** ve **"InsufficientFundException"** isimli iki farklı alt sınıf türetiniz. "NegativeAmountException" sınıfı, hem para yatırma hem de para çekme işlemlerinde negatif miktarlar üzerinde işlem yapılmaya çalışılması durumunda kullanılacaktır. "InsufficientFundException" sınıfı, hesaptan para çekme işlemi sırasında hesapta yeterli miktarda para olmaması durumunda kullanılacaktır.

* Hesapları temsil eden **"Account"** sınıfı aşağıdaki veri ve metotları içermelidir:

- **int id:** Hesap numarası karşılığı olarak
  double balance: Hesaptaki para miktarı karşılığı olarak
- Constructor metot(lar)
- Getter/Setter metotlar
- Parametre olarak geçirilen miktarı hesaptaki paraya ekleyecek **"deposit"** isimli metot.
- Parametre olarak geçirilen miktarı hesaptaki paradan düşecek **"withdraw"** isimli metot.
- **toString** metodu.
- **equals** metodu.
- **clone** metodu.

* Banka şubelerini temsil edecek **"BankOffice"** isimli sınıf aşağıdaki veri ve metotları içermelidir:

- **String bankaAd**: //ABC BANKASI gibi…
  String subeAd: //Bornova gibi …
  String subeAdres: // Ege Universitesi Kampus gibi…
  Account[] accounts: // Şubedeki tüm hesap nesnelerini tutan dizi
- Constructor metot(lar)
- Getter/Setter metotlar
- **toString** metodu
- **equals** metodu
- **clone** metodu

* **"TestSoru1"** isimli sınıf yazarak, bu sınıf içinde çeşitli hesap ve banka şubesi nesneleri oluşturarak, gerek Account gerekse BankOffice sınıfındaki tüm metotların (özellikle; deposit, withdraw, toString, equals ve clone metotları) kullanımını gösteriniz. Equals ve clone metotlarının kullanımını gösterdiğiniz komutların yanına oluşturmuş olduğunuz örnek nesnelere göre ne tür sonuçlar beklediğinizi yorum (// comment) olarak yazınız.

ÇÖZÜM:

```
public class AccountException extends Exception {
	public AccountException(String str) {
		super(str);
	}
}

public class NegativeAmountException extends AccountException {

	/** Construct an negative amount exception */
	public NegativeAmountException() {
		super("Negative Amount Cannot Be Processed");
	}
}

public class InsufficientFundException extends AccountException {

	/** Construct an insufficient exception */
	public InsufficientFundException() {
		super("Money Cannot Withdrawed: Insufficient Amount");
	}
}

public class Account implements Cloneable {
	// Two data fields in an account
	private int id;
	private double balance;

	/** Construct an account with specified id and balance */
	public Account(int id, double balance) {
		this.id = id;
		this.balance = balance;
	}

	/** Return id */
	public int getId() {
		return id;
	}
```

```
/** Setter method for balance */
public void setBalance(double balance) {
	this.balance = balance;
}

/** Return balance */
public double getBalance() {
	return balance;
}


/** Deposit an amount to this account */
public void deposit(double amount) throws NegativeAmountException {
	if (amount < 0)
		throw new NegativeAmountException();
	balance = balance + amount;
}

/** Withdraw an amount from this account */
public void withdraw(double amount) throws NegativeAmountException,
		InsufficientFundException {
	if (amount < 0)
		throw new NegativeAmountException();
	if (balance < amount)
		throw new InsufficientFundException();
	balance = balance - amount;
}

public String toString() {
	return "AccountId:" + id + "   " + "Balance" + balance;
}

public boolean equals(Object o) {
	if (o == this) {
		return true;
	} else if (o == null) {
		return false;
	} else if (o instanceof Account) {
		Account a = (Account) o;
		return getId() == a.getId() && getBalance() == a.getBalance();
	} else {
		return false;
	}
}
```

```
	public Object clone() {
		try {
			return super.clone();
		} catch (CloneNotSupportedException e) {
			e.getMessage();
			return null;
		}
	}
} // End of Account Class



public class Banka implements Cloneable {

	private String bankaAd;
	private String subeAd;
	private String subeAdres;
	private Account[] accounts;

	public Banka(String ad, String sad, String adres, Account[] accs) {
		bankaAd = ad;
		subeAd = sad;
		subeAdres = adres;
		accounts = accs;
	}

	public Account[] getAccounts() {
		return accounts;
	}

	public void setAccounts(Account[] accounts) {
		this.accounts = accounts;
	}

	public String getBankaAd() {
		return bankaAd;
	}

	public void setBankaAd(String bankaAd) {
		this.bankaAd = bankaAd;
	}

	public String getSubeAd() {
		return subeAd;
	}
```

```
public void setSubeAd(String subeAd) {
	this.subeAd = subeAd;
}

public String getSubeAdres() {
	return subeAdres;
}

public void setSubeAdres(String subeAdres) {
	this.subeAdres = subeAdres;
}

public String toString() {
	String deger = "BANKA AD:" + bankaAd + "  BANKA SUBESI:"
	+ subeAd     + "   SUBE ADRES:" + subeAdres + "\n" +
	"HESAPLAR:" + "\n";
	for (int i = 0; i < accounts.length; i++) {
		deger += accounts[i].toString();
		deger += "\n";
	}
	return deger;
}

public boolean equals(Object o) {
	if (o == this) {
		return true;
	} else if (o == null) {
		return false;
	} else if (o instanceof Banka) {
		Banka b = (Banka) o;
		if (bankaAd != b.getBankaAd() || subeAd != b.getSubeAd()
				|| subeAdres != b.getSubeAdres()) {
			return false;
		}
		Account[] myAccounts = this.getAccounts(), otherAccounts
					= b.getAccounts();
		for (int i = 0; i < myAccounts.length; i++) {
			if (!myAccounts[i].equals(otherAccounts[i])) {
				return false;
			}
			return true;
		}
	}
	return false;
}
```

```
	public Object clone() {
		try {
			Banka b = (Banka) super.clone();
			b.setAccounts(getAccountsCopy());
			return b;
		} catch (CloneNotSupportedException e) {
			System.out.println(e.getMessage());
			return null;
		}
	}

	public Account[] getAccountsCopy() {
		Account[] copy = (Account[]) accounts.clone();
		for (int i = 0; i < accounts.length; i++) {
			copy[i] = (Account) accounts[i].clone();
		}
		return copy;
	}
} // End of Class Bank


public class TestSoru1 {

	public static void main(String[] args) {

Account[] hesaplar = new Account[3];
		Account a1 = new Account(10, 5000);
		Account a2 = new Account(20, 14000);
		Account a3 = new Account(30, 8000);
		hesaplar[0] = a1;
		hesaplar[1] = a2;
		hesaplar[2] = a3;
		Banka bank1 = new Banka("Is Bankasi", "Bornova", "Kampus",
hesaplar);
		Banka bank2 = new Banka("Yapi Kredi", "Bornova", "Ata Duragi",
hesaplar);

		try {
			System.out.println("Account1:" + a1);
			System.out.println("Account2:" + a2);
			System.out.println("Account3:" + a3);
			a1.deposit(3000);
			System.out.println("Account1:" + a1);
			a1.withdraw(6000);
```

```
                System.out.println("Account1:" + a1);
                a1.withdraw(1000);
                System.out.println("Account1:" + a1);
                a1.withdraw(500);
                System.out.println("Account1:" + a1);
                System.out.println("BANKA:" + "\n" + bank1);
                System.out.println(a1.equals(a2));
                Account a4 = (Account) a1.clone();
                System.out.println(a1.equals(a4));
                System.out.println(bank1.equals(bank2));
                Banka bank3 = (Banka) bank1.clone();
                System.out.println(bank1.equals(bank3));
                System.out.println(bank3);
            } catch (NegativeAmountException e1) {
                System.out.println(e1.getMessage());
            } catch (InsufficientFundException e2) {
                System.out.println(e2.getMessage());
            }
        }

}
```

ARRAYLIST SINIFI

Otomatik olarak genişleyebilen diziler tanımlanabilir. Java 5'ten sonra gelen "Generics" özelliği ile ArrayList içinde tutulacak nesne tipleri de belirtilebilir.

Örneğin:

```
//Generic ArrayList to Store only String objects
ArrayList<String> stringList = new ArrayList<String>();
```

ArrayList Oluşturma ve Bazı Metotlarının Kullanımı:

```
import java.util.ArrayList;
import java.util.Iterator;

ArrayList<CorporatePerson> calisanlarList = new ArrayList<CorporatePerson>();
calisanlarList.add(yeniCalisan); // CorporatePerson tipinde olmalı
calisanlarList.remove(birCalisan); // CorporatePerson tipinde olmalı
int size=calisanlarList.size();
CorporatePerson cp=calisanlarList.get(i);
```

Liste üzerinde kullanılacak "Iterator" örnekleri "Tasarım Desenleri" anlatılan derslerde verilecektir.

ArrayList sınıfının diğer metotlarını da siz araştırınız…

“POLYMORPHIC TYPE ASSIGNMENT” KONUSU ÖRNEĞİ:

```
public abstract class T { …}

public interface A { … }

public class E extends T  implements A{…}

public class F extends T {…}

public class P extends E {…}
```

Yukarıda verilen sınıf sıradüzenini (hierarchy) göz önüne alarak, aşağıdaki çeşitli komut kullanımlarının doğru mu yoksa yanlış mı olduğunu DOĞRU, DERLEME-HATASI veya ÇALIŞMA-ZAMANI-HATASI anahtar kelimelerinden birisini kullanarak belirtiniz ve eğer hatalı ise nedenini açıklayınız.

i) `E e1=new F();`

ii) `T[] t2=new T[3];`

iii) `T t3=new E();`

iv) `A nesneE=new E();`

v) `A nesneP=new P();`

vi)
```
T t4=new P();
P p4= (P) t4;
```

vii)
```
T t5=new P();
F f5=(F)t5;
```