객체지향프로그래밍 -7주차 실습 활동지 (2019년 10월 15일)
성명: 안창희 학과: 소프트웨어 학번: 201723272 실습실명: 318호

※ 본 실습활동지를 작성함에 있어 다른 학생의 문서로부터 일부 또는 전체를 복사하였습니까
예( ) 아니오(O) (복사 하였다면 예에 체크하고 아니라면 아니오에 체크하시오)

1. 주어진 코드 (첨부파일 제공, Account, AccountTest 클래스)를 수행해 보고 다음 물음에 답하시오.

가) 다음과 같이 Account class의 subclass로 SavingAccount class를 작성하시오. (코드포함) (3점)

SavingAccount는 일정 이율을 가지고 있으며 이율에 따라 이자를 제공할 수 있다. 다음과 같이 이율은 객체 생성시 매개변수를 제공하여 constructor에서 초기화한다.

```
sa = new SavingAccount("Gildong", 10000.0, 4.5);
// 4.5 는 이율
```

balance에 이율을 곱해 이자를 지급하는 method (eg. addInterest)를 제공해야 한다. 이자는 balance에 더해진다.

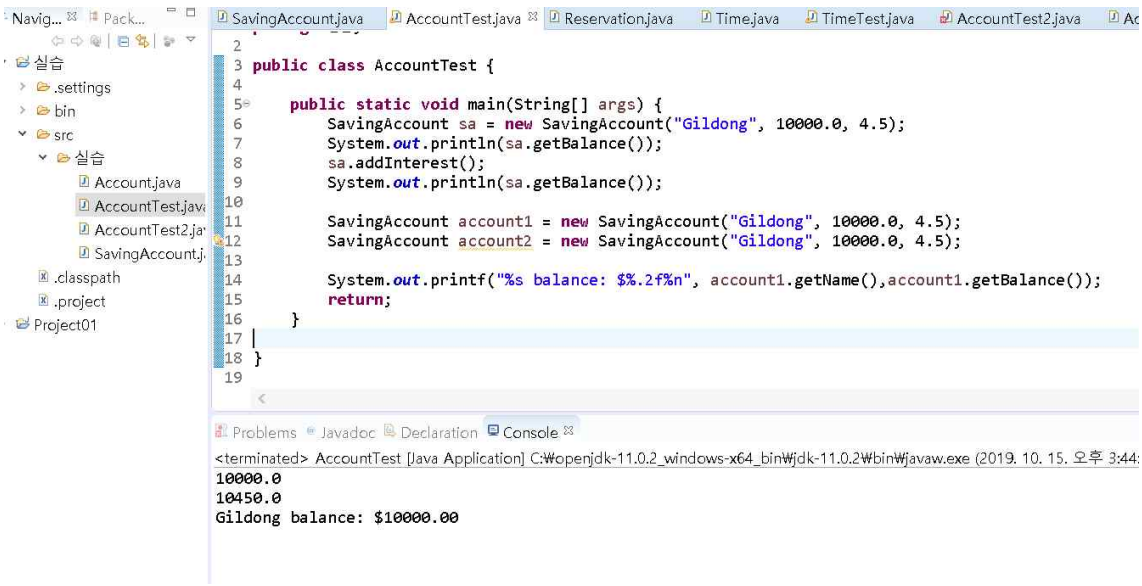
```
public void addInterest()  { ...  }
```

```
public class SavingAccount extends Account{
	double interest;

	public SavingAccount(String name, double balance, double interest) {
		super(name,balance);
		this.interest=interest;
	}
	public void addInterest() {
		deposit(this.getBalance() * (this.interest/100.0));
	}

	public String toString() {
		return (this.getName()+" balance: $" + this.getBalance());
	}

}
```

나) SavingAccount 클래스의 추가 기능을 테스트하는 AccountTest.java를 작성하시오. (코드와 실행화면포함) (2점)

힌트)
SavingAccount 객체 하나를 생성하여 balance를 확인하고 이자를 지급한 후 다시 balance를 확인하여 이자가 제대로 지급되어 있는지 테스트 함.

```
public class AccountTest {

	public static void main(String[] args) {
		SavingAccount sa = new SavingAccount("Gildong", 10000.0, 4.5);
		System.out.println(sa.getBalance());
		sa.addInterest();
		System.out.println(sa.getBalance());

		SavingAccount account1 = new SavingAccount("Gildong", 10000.0, 4.5);
		SavingAccount account2 = new SavingAccount("Gildong", 10000.0, 4.5);

		System.out.printf("%s balance: $%.2f%n",
account1.getName(),account1.getBalance());
		return;
	}

}
```



==========

다) 위 AccountTest에서 Account 정보를 다음과 같이 출력하고 있다.

System.out.printf("%s balance: $%.2f%n", account1.getName(),account1.getBalance());

위 문장을 다음 같이 변경하여도 동일하게 출력할 수 있도록 Account class에 toString을 추가하시오. 단, balance값의 소숫점이하 자리수는 일치하지 않아도 됨. (코드 포함) (3점)

System.out.println(account1);

==========

```
public class Account {

	private String name; // instance variable
	private double balance; // instance variable

	public Account(String name, double balance) {
		this.name = name; // assign name to instance variable name

		if (balance > 0.0) // if the balance is valid
			this.balance = balance; // assign it to instance variable balance
	}

	public void deposit(double depositAmount) {
		if (depositAmount > 0.0) // if the depositAmount is valid
			balance = balance + depositAmount; // add it to the balance
	}

	public double getBalance() {
		return balance;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public String toString() {
		return (this.getName()+" balance: $" + this.getBalance());
	}
```

```
} // end class Account
```

==========

라) 다음 코드 AccountTest2를 실행한 후 각 결과 값이 왜 그런지 설명하시오. (2점)

```
public class AccountTest2 {

	public static void main(String[] args) {
		Account account1 = new Account("Jane Green", 1000.0);
		Account account2 = account1;
		Account account3 = new Account("Jane Green", 1000.0);
		System.out.println(account1 == account2); //1
		System.out.println(account1.equals(account2));  //2
		System.out.println(account1 == account3); //3
		System.out.println(account1.equals(account3)); //4
	}

}
```

==========

위의 코드에서 Account 형 객체는 총 2개가 생성이 되었으며, 이를 가리키는 객체 변수는 총 3개이다. account1과 account2는 첫 번째로 생성된 객체를 동시에 지칭하며 account1 == account2문을 사용하든 equals() 메소드를 통해서 지칭하는 객체를 비교하든 동일한 객체를 지칭하는 상태에서 이 둘을 비교하므로 무조건 true가 출력될 수 밖에 없다.

하지만 account3는 객체안의 데이터는 동일하지만 두 번째 생성된 "다른" 객체이므로 첫 번째 생성된 객체와 비교하여 ==문을 사용하든, equals() 메소드를 사용하든 false가 출력된다.

==========

마) Account 객체간의 비교를 객체의 상태값에 따라 비교할 수 있도록 Account class에 equals method를 override하시오. (코드포함) (3점)

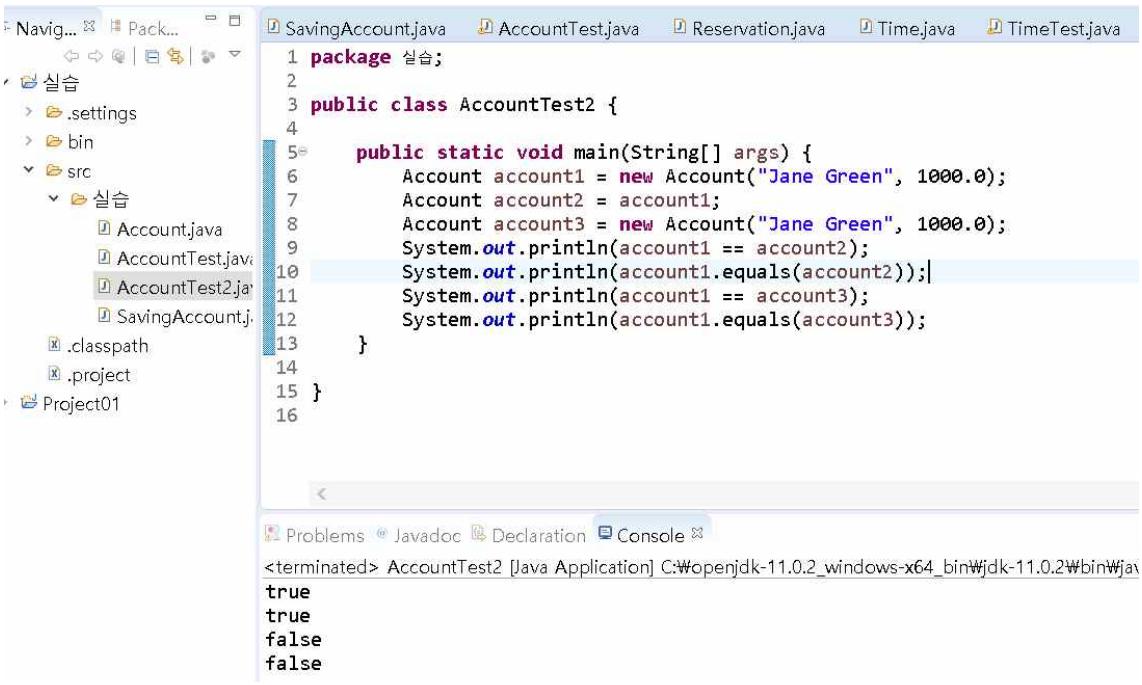
==========

```
public class Account {

	private String name; // instance variable
	private double balance; // instance variable

	public Account(String name, double balance) {
		this.name = name; // assign name to instance variable name

		if (balance > 0.0) // if the balance is valid
			this.balance = balance; // assign it to instance variable balance
	}

	public void deposit(double depositAmount) {
		if (depositAmount > 0.0) // if the depositAmount is valid
			balance = balance + depositAmount; // add it to the balance
	}

	public double getBalance() {
		return balance;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}


	public boolean equals(Account b) {
		if (this.getName() == b.getName()) {
			if (this.getBalance() == b.getBalance()) {
				return true;
			}
		}
		return false;
	}
```

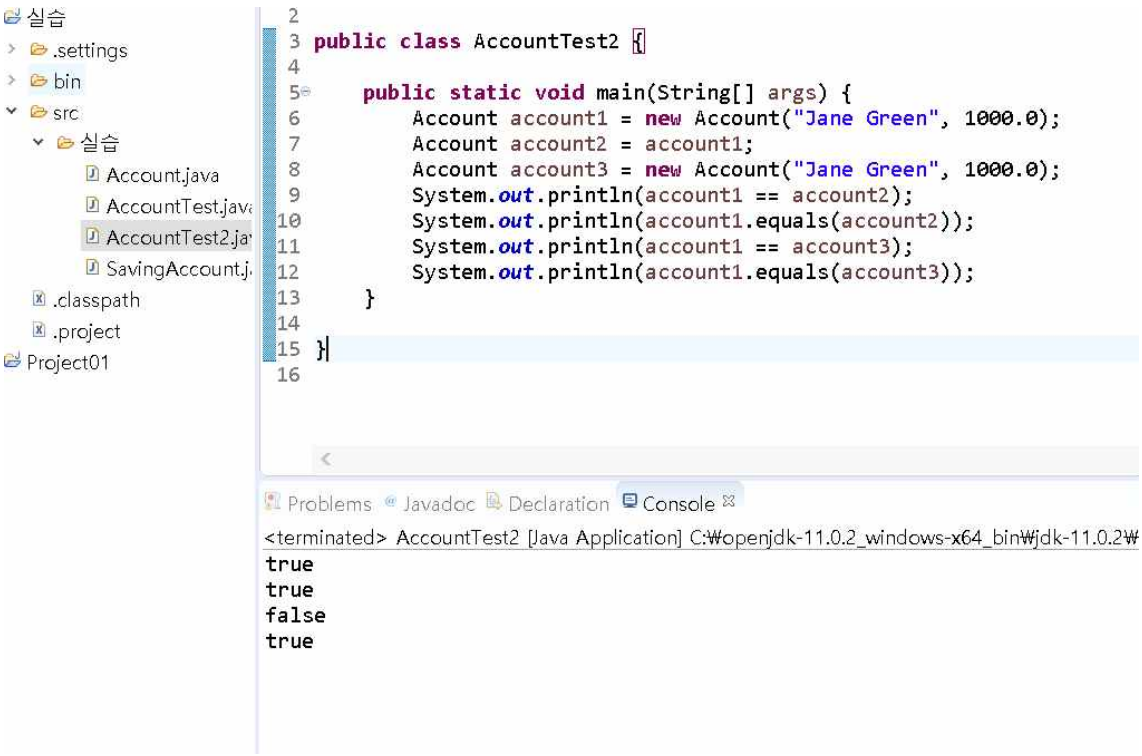
```
} // end class Account
================================================================
```

바) (라)번의 AccountTest2를 수행한 후 결과를 비교한 후 차이에 대하여 설명하시오.
(실행결과, 설명포함) (2점)

==========



(바) 의 결과



(마) 의 결과

```
System.out.println(account1 == account2); //1
System.out.println(account1.equals(account2));   //2
System.out.println(account1 == account3); //3
System.out.println(account1.equals(account3)); //4
```

(바)의 코드에서== 비교문과 equals() 메소드를 통해서 지칭하는 객체를 비교하게 되면 객체 자체가 완전 동일할때만 true가 출력이 되므로 //1 과 //2 의 문장만 true가 출력이 되지만

(마)의 코드에서는 Account 클래스에 적용되는 equals 메소드를 override 하였으므로 객체 자체가 아닌, 본인이 정의한대로 Account 객체 안의 name과 balance가 일치할때만 true를 반환하는 함수로 동작하기 때문에 //3 에서도 또한 true가 출력되게 동작한다.

==========