Com S 228, Spring 2018, Exam 1 Key

1.

`MoneyMarketAccount sharpay =`
`  new MoneyMarketAccount("Sharpay␣Evans", 1,`
`                         2000000.00, 4.25);`
`System.out.println(sharpay.getInterestRate());`
	4.25

`Account troy =`
`  new SavingsAccount("Troy␣Bolton", 2,`
`                     3000.00, 1.125);`
`System.out.println(troy.getBalance());`
	3000

`Account troy =`
`  new SavingsAccount("Troy␣Bolton", 2,`
`                     3000.00, 1.125);`
`troy.withdraw(5000);`
`System.out.println(troy.getBalance());`
	IllegalArgumentException

`Account kelsi =`
`  new SavingsAccount("Kelsi␣Nielsen", 3,`
`                     500.00, 1.125);`
`System.out.println(kelsi.getAccountInfo());`
	Savings: Kelsi Nielsen, 3

`CheckingAccount darbus =`
`  new MoneyMarketAccount("Mrs.␣Darbus", 4,`
`                         10000.00, 4.25);`
`darbus.calculateInterest(365);`
	No calculateInterest() in class CheckingAccount.

`InterestBearing chad =`
`  new CheckingAccount("Chad␣Danforth", 5,`
`                      777.77);`
`System.out.println(chad.getInterestRate());`
	Cannot convert CheckingAccount to InterestBearing.

`InterestBearing ryan =`
`  new SavingsAccount("Ryan␣Evans", 6,`
`                     400000.00, 1.125);`
`System.out.println(ryan.getInterestRate());`
	1.125

`InterestBearing gabriella =`
`  new MoneyMarketAccount("Gabriella␣Montez",`
`                         10000, 7, 4.25);`
`System.out.println(`
`  ((MoneyMarketAccount)`
`   gabriella).getAccountInfo());`
	Money Market: Gabriella Montez, 10000

`CheckingAccount taylor =`
`  new CheckingAccount("Taylor␣McKessie", 8,`
`                      7000.00);`
`System.out.println(`
` ((InterestBearing) taylor).getInterestRate());`
	ClassCastException

2. a)
```
@Override
public Object clone()
{
  try {
    Complex c = (Complex) super.clone();
    return c;
  }

  catch (CloneNotSupportedException e) {
    return null;
  }
}
```

b)
```
@Override
public boolean equals(Object o)
{
  if (o == null || o.getClass() ! = getClass()) {
    return false;
  }

  // typecast o to Complex so that we can compare data members
  ComplexTuple t = (ComplexTuple) o;
  // Compare the data members and return accordingly
  return c1.equals(t.c1) && c2.equals(t.c2);
}
```

Or

```
@Override
public boolean equals(Object o)
{
  // If the object is compared with itself then return true
  if (o == this) {
    return true;
  }

  /* Check if o is an instance of Complex or not
  * "null instanceof [type]" also returns false */
  if (!(o instanceof ComplexTuple)) {
    return false;
  }

  // typecast o to Complex so that we can compare data members
  ComplexTuple t = (ComplexTuple) o;
  // Compare the data members and return accordingly
  return c1.equals(t.c1) && c2.equals(t.c2);
}
```

3. a)
    i) $O(n)$
    ii) $O(n)$
    iii) $O(n^2)$
   b)
    i) $O(\log n)$
    ii) $O(n^2)$
    iii) $O(n^2 \log n)$
    iv) $O(n^3)$
   c)
    i) $O(n)$
    ii) $O(n)$
   d) $O(n \log n)$

4. a) Insertion Sort
   b) Merge Sort
   c) Quick Sort
   d) Selection Sort
   e) 0 1 4 6 7 5 3 2
   f) 1 4 6 7 0 2 3 5