1.

```
package Proj08p1;

public class A {
	public int value;

	public A() {   //construtor
	   this.value = 0;
	}

	public A(int val1) {   // construtor
	   this.value=val1;
	}

	public A(A a) {   // construtor
	   this.value= getValue(a);
	}

	public int getValue(A a) {
	   return a.value;
	}

	/**
	 * Verify if the numbers have the same value
	 * @param a
	 * @return
	 */
	public boolean equals(A a){
	   return (this.value == a.value)?(true):(false);
	}
}
```

```
package Proj08p1;

public class B {
	private int value;

	public B(int val2) {
	   this.value =val2;
	}

	public int getValue() {
	   return this.value;
	}
}
```

```
package Proj08p1;

public class C {
	public int target;

	public void verify(B b) {   //construtor
		this.target = b.getValue();
	}

	/**
	 * @return highest digit of the number
	 */
	public int getHighestDigit() {
	   int number = this.target;
	   int highest = number % 10;    //resto da divisão por 10 para comparar dígito a
dígito
	   do{
	       number /= 10;
	       int res = number % 10;
	       if (res > highest)
```

```
                highest = res;
            } while (number >0);
            return highest;
        }

        /**
         * verify if the number is pair or not
         * @return true if it's a pair number or false if it's a odd number
         */
        public boolean isPair() {
                return (target%2==0)?(true):(false);
        }
}
```

```
public class App {
    public static void main(String[] args) {
        A a = new A();
        a.value = 24;

        B b = new B(a.value);
        System.out.println(b.getValue());

        C c = new C();
        c.verify(b);
        System.out.println(c.target);
        System.out.println(c.isPair());
        System.out.println(c.getHighestDigit());

        A a2 = new A(a);
        A a3 = new A(9);
        System.out.println(a2.equals(a));
        System.out.println(a3.equals(a));
        System.out.println(a3.value);
         }
    }
```

Output:

```
24
24
true
4
true
false
9
```

2.

```
package proj08p2;

/**
 * This class objects represent selling candy machines.
 * Two types of service are available: Basic and premium
 * @author FilipaG
 *
 */
public class CandyDispenser {
    private final int maxCandyUnits;    //capacidade máxima da máquina de rebuçados
    private final int numCandyBasic;           // número de rebuçados do serviço base
    private final int numCandySpecial;          //número de rebuçados do serviço especial
    private final int candyCost;         // preço dos rebuçados
    private int stock;     //quantidade de rebuçados na máquina
    private int salesMoney;     //dinheiro no depósito de moedas
    private int extractedMoney;    //dinheiro já extraído do depósito de moedas
    private int record;         //máximo valor obtido em vendas em cada extração monetária
```

```
    public CandyDispenser(int maxUnits, int basic, int special, int candyCost) {
//construtor
        this.maxCandyUnits = maxUnits;
        this.numCandyBasic = basic;
        this.numCandySpecial = special;
        this.candyCost = candyCost;
        this.stock = maxUnits;
    }

/**
 * Provides a basic service of candies, if you have enough "stock" .
 * If there is no "stock" the request is ignored and the payment should not be considered
 */
        public void serveBasic()
        {
            if (stock >= numCandyBasic)
            {
                stock -= numCandyBasic;  //redução no stock da quantidade de rebuçados
vendidos com o serviço básico
                salesMoney += numCandyBasic * candyCost;  // Ganho (em cêntimos) com a venda
de rebuçados
            }
        }

/**
 * Provides a special service of candies , if you have enough "stock".
 * If there is no "stock" the request is ignored and the payment should not be considered
 */
        public void serveSpecial()
        {
            if (stock >= numCandySpecial)
            {
                stock -= numCandySpecial;  //redução no stock da quantidade de rebuçados
vendidos com o serviço especial
                salesMoney += numCandySpecial * candyCost;  // Ganho (em cêntimos) com a
venda de rebuçados
            }
        }

/**
 * Indicates the total amount of sales that the machine ever made since the last time it
 * was withdrawn money from the deposit of coins .
 */
        public int getValueSold()
        {
            return salesMoney;      //valor das vendas desde a última extração de moedas
        }

/**
 * Indicates the value of total sales of existing candies on deposit.
 * @return
 */
        public int getValueOnSale()
        {
            return stock * candyCost;            // valor (em cêntimos) do stock
        }

/**
 * Extracts of the machine all the accumulated money in the coin hopper .
 */
        public void extractMoney()
        {
            extractedMoney += salesMoney;
            if (extractedMoney>record)
                record = extractedMoney; //verificação se se trata de um novo record de
vendas
            salesMoney = 0 ;      // o valor no depósito de moedas volta a zero
```

```
        }

/**
 * Adds more units to deposit candy mints. The total number of candies in the deposit can
 * not exceed the limit specified when the machine was created . If you try to enter a
 * higher value , the excess should be ignored .
 * @param units
 */
        public void reFill(int units)
        {
           if (stock + units > maxCandyUnits)   //se a quantidade de rebuçados em stock
com a recarga for superior ao máximo da capacidade da máquina, ela fica cheia e o excesso
é ignorado
               stock = maxCandyUnits;
           else
               stock += units;
        }

/**
 * Indicates the highest amount of money ever the machine contained in the tank , since
it was put into operation .
 * @return
 */
        public int sellingRecord()
        {
           return record;
        }
```

```
        public static void main(String[] args) {
           CandyDispenser c = new CandyDispenser(20,2,4,2);

           c.serveBasic();    // stock = 18; salesMoney = 2*2=4
           c.serveBasic();    // stock = 16; salesMoney = 4 + 2*2 = 8
           c.reFill(30);       // stock = 20
           c.serveBasic();     // stock = 18; salesMoney = 8 + 2*2 = 12
           c.serveSpecial();   // stock = 14; salesMoney = 12 + 4*2 = 20
           c.serveSpecial();   // stock = 10; salesMoney = 20 + 4*2 = 28
           System.out.println("Value On Sale:" + c.getValueOnSale());    // 10 * 2 = 20
           System.out.println("Value Sold:" + c.getValueSold()); //salesMoney = 28
           c.serveBasic();        // stock = 8; salesMoney = 28 + 2*2 = 32
           c.getValueOnSale();    // 8 * 2 = 16(int)
           System.out.println("Value Sold:"+ c.getValueSold());     // salesMoney = 32
           c.extractMoney();     // extractedMoney = 32; Record = 32;  salesMoney = 0;
           c.serveSpecial();     // stock = 4; salesMoney = 4*2 = 8
           c.serveSpecial();      //stock = 0; salesMoney = 8 + 4*2 = 16
           System.out.println("Value Sold:" + c.getValueSold());    //salesMoney = 16(int)
           c.reFill(100);          // stock = 20
           System.out.println("Value On Sale:"+ c.getValueOnSale());     //20 * 2= 40(int)
           System.out.println("Value Sold:" + c.sellingRecord());     //  Record = 32;
        }
}
```

Output:

```
Value On Sale:20
Value Sold:28
Value Sold:32
Value Sold:16
Value On Sale:40
Value Sold:32
```

```
//Imprimi só para verificar se os valores correspondiam ao que esperava
```