```java
package uebung101;

import java.util.concurrent.Semaphore;

class Output extends Thread {

    public void run() {
                try {
                    InOut.getSema().acquire();
                    if (InOut.isEntered()) {
                        System.out.println(InOut.
getValue() * InOut.getValue());}
                } catch (InterruptedException e) {
                }
                finally{
                    InOut.getSema().release();
                }
            }


}

class Input extends Thread {

    public void run() {
        try {
            InOut.getSema().acquire();
            InOut.setValue(IO.readInt("Value: "));
            InOut.setEntered(true);
        } catch (InterruptedException e) {
        }
        finally{
            InOut.getSema().release();
        }

    }
}

public class InOut {

    private static Semaphore sema = new Semaphore(1
```

```
40 );
41     private static boolean entered = false;
42     private static int value = 0;
43 
44     public static Semaphore getSema() {
45         return sema;
46     }
47 
48     public static boolean isEntered() {
49         return entered;
50     }
51 
52     public static void setEntered(boolean entered
   ) {
53         InOut.entered = entered;
54     }
55 
56     public static int getValue() {
57         return value;
58     }
59 
60     public static void setValue(int value) {
61         InOut.value = value;
62     }
63 
64     public static void main(String[] args) {
65 
66         new Input().start();
67         new Output().start();
68 
69     }
70 
71 
72 }
```

```java
package uebung102;

import java.util.concurrent.Semaphore;

public class Barriers {

    private final static int NUMBER = 3;

    public static void main(String[] args) {
        NumberRunner[] runner = new NumberRunner[NUMBER];
        for (int i = 0; i < NUMBER; i++) {
            runner[i] = new NumberRunner(i);
        }
        for (int i = 0; i < NUMBER; i++) {
            runner[i].start();


        }
    }

}
class NumberRunner extends Thread {

    private int number;
    private Semaphore barrier = new Semaphore(10);

    public NumberRunner(int n) {
        number = n;

    }

    @Override
    public void run() {
        for (int i = 0; i < 1000; i++) {
            System.out.println("Thread " + number + ": " + i);
            if ((i + 1) % 10 == 0) {
                try {
                    barrier.acquire();
                    barrier.release();
```

```
40                  } catch (InterruptedException e) {
41                      e.printStackTrace();
42                  }
43              }
44          }
45      }
46  }
47
```

```java
package uebung103;

public class Item {

    private String name;
    private int value;
    private int weight;

    public Item(String name, int value, int weight) {
        super();
        this.name = name;
        this.value = value;
        this.weight = weight;
    }

    public String getName() {
        return name;
    }

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

    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
    }

    public int getWeight() {
        return weight;
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }

    @Override
```

```
41      public String toString() {
42          return name;
43      }
44
45      @Override
46      public int hashCode() {
47          final int prime = 31;
48          int result = 1;
49          result = prime * result + ((name == null
    ) ? 0 : name.hashCode());
50          result = prime * result + value;
51          result = prime * result + weight;
52          return result;
53      }
54
55      @Override
56      public boolean equals(Object obj) {
57          if (this == obj)
58              return true;
59          if (obj == null)
60              return false;
61          if (getClass() != obj.getClass())
62              return false;
63          Item other = (Item) obj;
64          if (name == null) {
65              if (other.name != null)
66                  return false;
67          } else if (!name.equals(other.name))
68              return false;
69          if (value != other.value)
70              return false;
71          if (weight != other.weight)
72              return false;
73          return true;
74      }
75
76  }
77
```

```java
package uebung103;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

public abstract class Knapsack {

    protected List<Item> candidates = new ArrayList<>();
    protected int capacity;

    public Knapsack(int capacity, Collection<Item> candidates) {
        super();
        this.capacity = capacity;
        this.candidates.addAll(candidates);
    }

    public int getCapacity() {
        return capacity;
    }

    public void setCapacity(int capacity) {
        this.capacity = capacity;
    }

    public List<Item> getCandidates() {
        return candidates;
    }

    public abstract Selection pack();

    private static final int REPETITIONS = 5;
    private static final int CAPACITY = 49;

    public static void main(String[] args) {
        List<Item> items = new ArrayList<>();
        items.add(new Item("Banknote", 100, 1));
        items.add(new Item("Goldbar", 1000, 30));
        items.add(new Item("Diamond", 750, 5));
```

```
40          test("Recursive", new KnapsackRecursive(
   CAPACITY, items));
41          test("Greedy", new KnapsackGreedy(CAPACITY
   , items));
42          test("Dynamic Programming", new
   KnapsackDynamic(CAPACITY, items));
43      }
44
45      private static void test(String title, Knapsack
    knapsack) {
46          System.out.print(title);
47          Selection result = null;
48          long totalNs = 0;
49          for (int i = 0; i < REPETITIONS; i++) {
50              long start = System.nanoTime();
51              result = knapsack.pack();
52              long stop = System.nanoTime();
53              totalNs += stop - start;
54              System.out.print(".");
55          }
56          System.out.println("\n\t" + result);
57          totalNs /= REPETITIONS;
58          long totalMs = totalNs / 1000000L;
59          System.out.println("\tTime required: " +
   totalNs + " ns (~ " + totalMs + " ms)");
60          System.out.println();
61      }
62
63 }
64
```

```java
package uebung103;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Selection {

    private Map<Item, Integer> items = new HashMap
<>();
    private int value;
    private int weight;

    public Selection() {
        super();
    }

    public Selection(Selection previous) {
        super();
        items.putAll(previous.items);
        value = previous.value;
        weight = previous.weight;
    }

    public Selection(Selection previous, Item item
) {
        super();
        items.putAll(previous.items);
        value = previous.value;
        weight = previous.weight;
        add(item);
    }

    public void add(Item item) {
        items.put(item, getCount(item) + 1);
        value += item.getValue();
        weight += item.getWeight();
    }

```

```
40      public int getCount(Item item) {
41          Integer result = items.get(item);
42          if (result == null) {
43              result = 0;
44          }
45          return result;
46      }
47
48      public Collection<Item> getItems() {
49          return items.keySet();
50      }
51
52      public int getValue() {
53          return value;
54      }
55
56      public void setValue(int value) {
57          this.value = value;
58      }
59
60      public int getWeight() {
61          return weight;
62      }
63
64      public void setWeight(int weight) {
65          this.weight = weight;
66      }
67
68      @Override
69      public String toString() {
70          StringBuilder b = new StringBuilder();
71          b.append("Value: ");
72          b.append(value);
73          b.append(", weight: ");
74          b.append(weight);
75          b.append(", items: ");
76          List<Item> list = new ArrayList<>(items.
    keySet());
77          list.sort((i1, i2) -> i1.getName().
    compareTo(i2.getName()));
78          for (int i = 0; i < list.size(); i++) {
```

```
            b.append(items.get(list.get(i)));
            b.append("x ");
            b.append(list.get(i).getName());
            if (i < list.size() - 1) {
                b.append(", ");
            }
        }
        return b.toString();
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((items == null
) ? 0 : items.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Selection other = (Selection) obj;
        if (items == null) {
            if (other.items != null)
                return false;
        } else if (!items.equals(other.items))
            return false;
        return true;
    }

}

```

```java
package uebung103;

import java.util.*;

public class KnapsackGreedy extends Knapsack {

    public KnapsackGreedy(int capacity, Collection<Item> candidates) {
        super(capacity, candidates);
    }

    @Override
    public Selection pack() {
        //TODO: implement this
        // Erstelle eine separate Liste für die sortierten Kandidaten
        List<Item> sortedCandidates = new ArrayList<>(getCandidates());

        // Verwende einen Comparator, um die Liste nach dem Gewicht aufsteigend zu sortieren
        Collections.sort(sortedCandidates, new Comparator<Item>() {
            @Override
            public int compare(Item item1, Item item2) {
                return Integer.compare(item1.getWeight(), item2.getWeight());
            }
        });
        Selection bestSelection = new Selection();
        for (Item item : sortedCandidates) {
            // Berechne die maximale Anzahl von Elementen, die in den Rucksack passen
            int maxItemCount = (getCapacity() - bestSelection.getWeight()) / item.getWeight();

            // Füge das Element die maximale Anzahl an Malen hinzu
            for (int count = 0; count < maxItemCount; count++) {
```

```
31                  bestSelection.add(item);
32              }
33          }
34          return new Selection();
35      }
36
37  }
38
```

```java
package uebung103;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

public class KnapsackDynamic extends Knapsack {

    public KnapsackDynamic(int capacity, Collection<Item> candidates) {
        super(capacity, candidates);
    }

    @Override
    public Selection pack() {
        //TODO: implement this


        return new Selection();
    }

}

```

```java
package uebung103;

import java.util.Collection;
import java.util.List;

public class KnapsackRecursive extends Knapsack {

    public KnapsackRecursive(int capacity, Collection<Item> candidates) {
        super(capacity, candidates);
        this.candidates.sort((i1, i2) -> Integer.compare(i1.getValue(), i2.getValue()));
    }

    @Override
    public Selection pack() {
        Selection bestSelection = new Selection();
        recursivePack(0, new Selection(), bestSelection);
        return bestSelection;
    }

    private void recursivePack(int index, Selection currentSelection, Selection bestSelection) {
        if (currentSelection.getWeight() > getCapacity()) {
            return; // Abbruch, da Gewicht überschritten
        }

        if (currentSelection.getValue() > bestSelection.getValue()) {
            bestSelection = new Selection(currentSelection); // bessere Auswahl gefunden
        }

        if (index >= getCandidates().size()) {
            return; // Abbruch, alle Gegenstände betrachtet
        }

```

```
33          Item currentItem = getCandidates().get(
    index);
34          int maxItemCount = (getCapacity() -
    currentSelection.getWeight()) / currentItem.
    getWeight();
35
36          for (int count = 0; count <= maxItemCount;
    count++) {
37              Selection newSelection = new Selection(
    currentSelection, currentItem);
38              recursivePack(index + 1, newSelection,
    bestSelection);
39          }
40
41      }
42  }
43
```