```java
public User findUser(User user) {
    int index = users.indexOf(user);
    if (index != -1) {
        return users.get(index);
    } else {
        return null;
    }
}
```

Users use KWArrayList and its indexOf method is O(n). It can be in the first index or never exists. Best case constant and worst case linear. Get method is constant time. So total is O(n).

```java
public void addUser(User newUser) {
    try {
        users.add(newUser);
    } catch (NullPointerException e) {
        System.err.println("User object reference is null!");
    }
    if(newUser instanceof Customer) {
        numOfCustomers++;
    }
}
```

Adding user to KWArrayList is amortized constant time. Other lines are also constant. Total is θ(1).

```java
public User removeUser(User userToBeRemoved) {
    User user = null;
    for (int i = 0; i < users.size(); ++i) {
        user = users.get(i);
        if (user == userToBeRemoved) {
            users.remove(i);
            return user;
        }
    }
    return user;
}
```

Getting user with specified index is constant (since uses KWArrayList), removing user from index is O(n). It can be last item or first item in the list. It needs n times shifting if it is the first item. Constant if it is the last item. Whole method is θ(n), since if it is in the beginning it needs n times shifting. If it is in the last it means it already iterated n times but removing is constant. So total is linear all the time.

```java
public Branch removeBranch(Branch branchToBeRemoved) {
    Branch branch = null;
    ListIterator<Branch> iter = branches.listIterator();
    while (iter.hasNext()){
        branch = (Branch) iter.next();
        if (branch == branchToBeRemoved) {
            iter.remove();
            return branch;
        }
    }
    return branch;
}
```

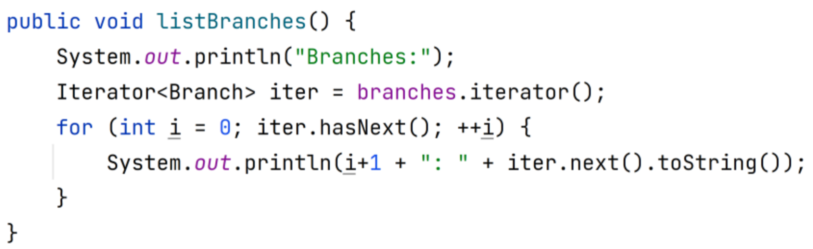
It is simply O(n). Iterator iterates over the linked list till reaching the specified branch. It may be in the first node or last node. Best case it is constant, worst case is linear. Removing element with iterator is constant.

```java
public User findUser(String email, String password) {
    UnknownUser aUser = new UnknownUser(email, password);
    User foundUser = null;
    for (int i = 0; i < users.size(); ++i) {
        foundUser = users.get(i);
        if (foundUser.equals(aUser)) {
            return foundUser;
        }
    }
    return foundUser;
}
```

It is O(n). In best case it founds the user at first index. Worst case is it never finds the user, so iterates whole array. Getting element from array list is constant. Equals method is constant.

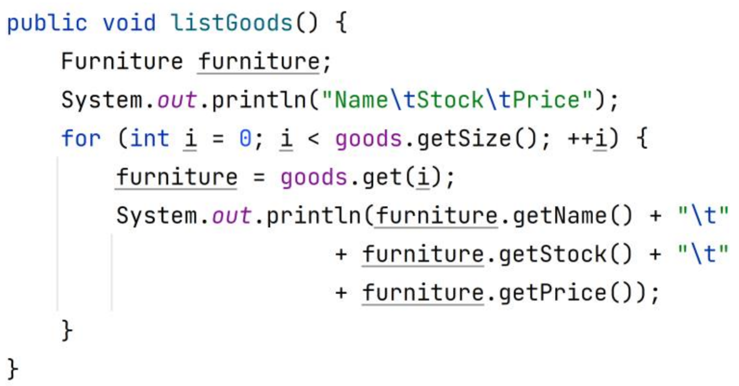
```java
public boolean isBranch(Branch branch) {
    Iterator<Branch> iter = branches.iterator();
    while (iter.hasNext()){
        if (iter.next() == branch) {
            return true;
        }
    }
    return false;
}
```

It is O(n). In best case it is the first node, worst case it never occurs.



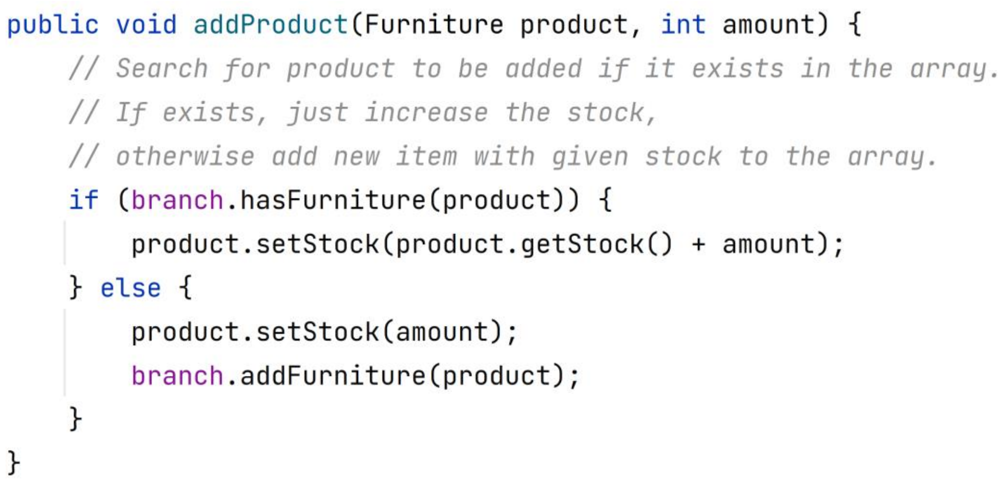
```java
public void listBranches() {
    System.out.println("Branches:");
    Iterator<Branch> iter = branches.iterator();
    for (int i = 0; iter.hasNext(); ++i) {
        System.out.println(i+1 + ": " + iter.next().toString());
    }
}
```

It is θ(n). It always iterates over the whole list and prints the data.



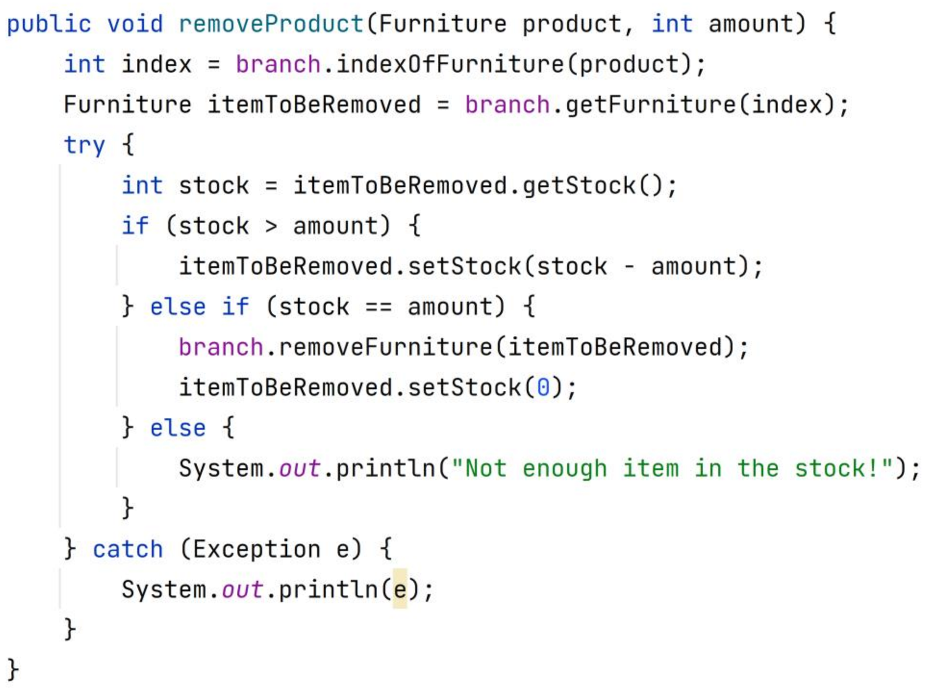
```java
public void listGoods() {
    Furniture furniture;
    System.out.println("Name\tStock\tPrice");
    for (int i = 0; i < goods.getSize(); ++i) {
        furniture = goods.get(i);
        System.out.println(furniture.getName() + "\t"
                + furniture.getStock() + "\t"
                + furniture.getPrice());
    }
}
```

Furnitures use HybridList and HybridList's get method is O(n), since it initializes a list iterator with given index to find the correct ArrayList node. Getting an element from this ArrayList is constant. Total is O(n^2). It iterates over whole list and loop happens n times. Θ(n)*O(n) = O(n^2)



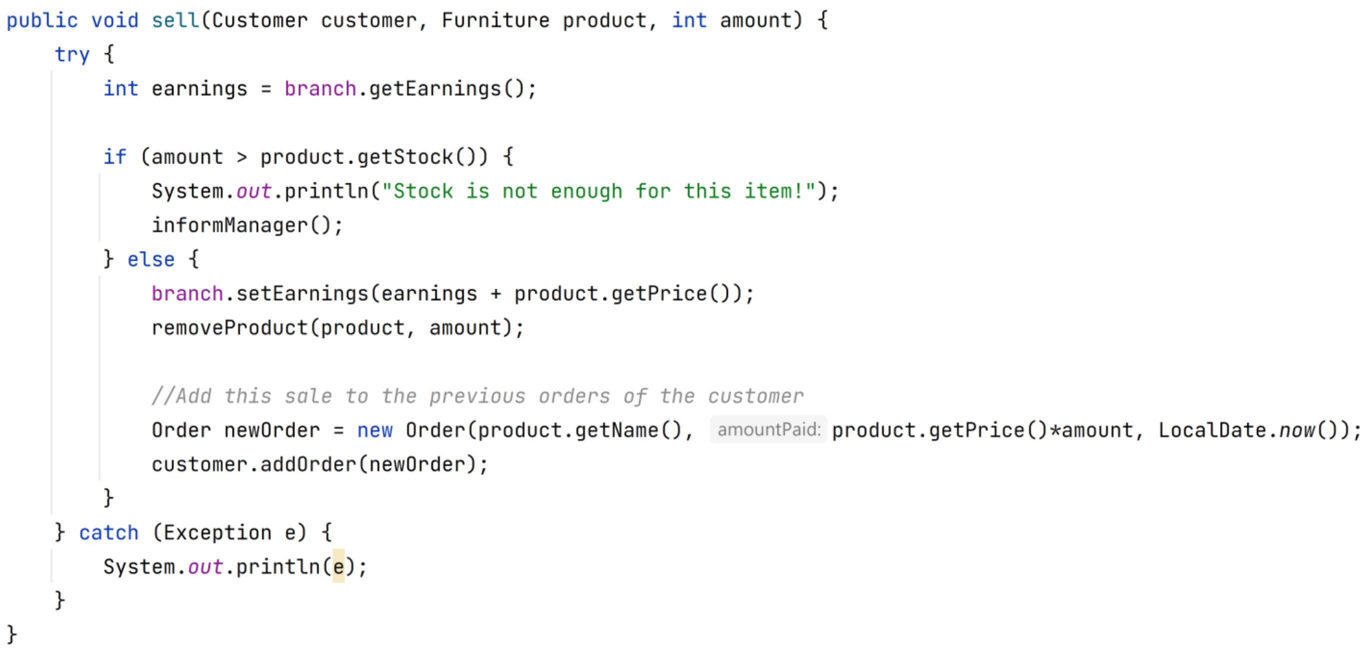
```java
public void addProduct(Furniture product, int amount) {
    // Search for product to be added if it exists in the array.
    // If exists, just increase the stock,
    // otherwise add new item with given stock to the array.
    if (branch.hasFurniture(product)) {
        product.setStock(product.getStock() + amount);
    } else {
        product.setStock(amount);
        branch.addFurniture(product);
    }
}
```

Has furniture is O(n). The rest is constant. Total is O(n).



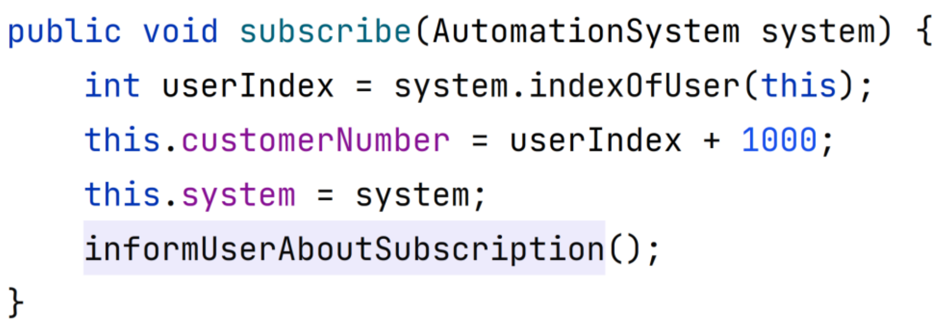
```java
public void removeProduct(Furniture product, int amount) {
    int index = branch.indexOfFurniture(product);
    Furniture itemToBeRemoved = branch.getFurniture(index);
    try {
        int stock = itemToBeRemoved.getStock();
        if (stock > amount) {
            itemToBeRemoved.setStock(stock - amount);
        } else if (stock == amount) {
            branch.removeFurniture(itemToBeRemoved);
            itemToBeRemoved.setStock(0);
        } else {
            System.out.println("Not enough item in the stock!");
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Index of furniture is O(n). getFurniture is O(n). removeFurniture is O(n). Total is O(n).



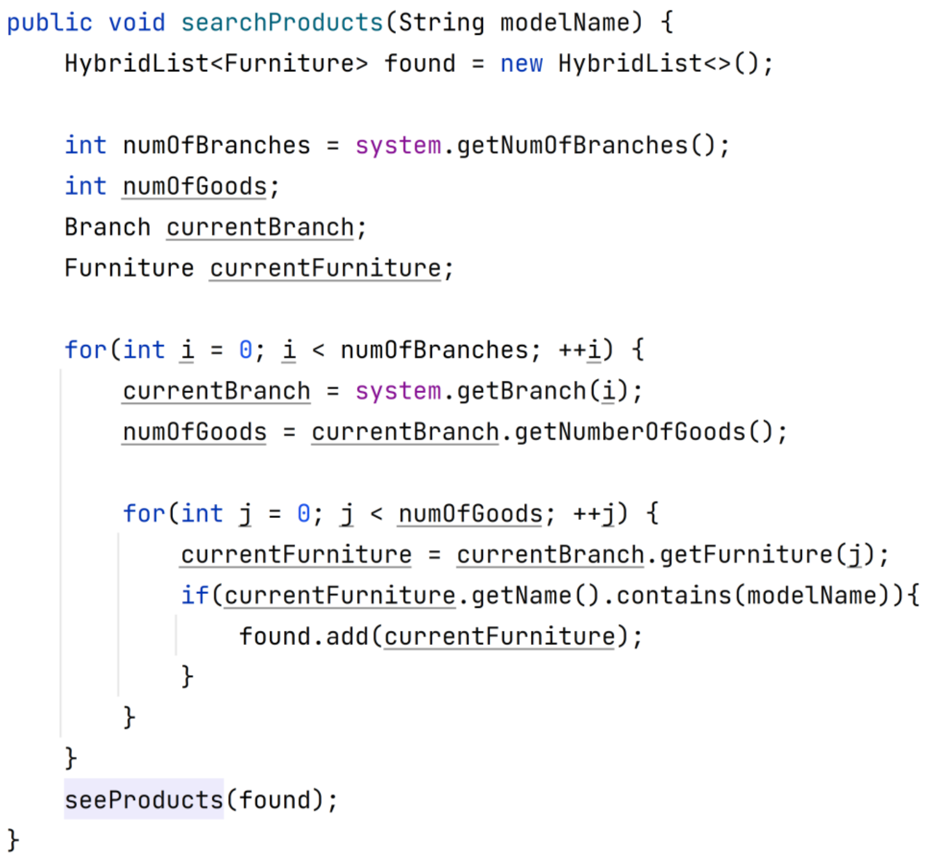
```java
public void sell(Customer customer, Furniture product, int amount) {
    try {
        int earnings = branch.getEarnings();

        if (amount > product.getStock()) {
            System.out.println("Stock is not enough for this item!");
            informManager();
        } else {
            branch.setEarnings(earnings + product.getPrice());
            removeProduct(product, amount);

            //Add this sale to the previous orders of the customer
            Order newOrder = new Order(product.getName(), amountPaid: product.getPrice()*amount, LocalDate.now());
            customer.addOrder(newOrder);
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Remove product is O(n). Orders uses array list, so adding an element to orders is amortized constant time. Total is O(n).



```java
public void subscribe(AutomationSystem system) {
    int userIndex = system.indexOfUser(this);
    this.customerNumber = userIndex + 1000;
    this.system = system;
    informUserAboutSubscription();
}
```

indexOfUser is O(n) time. Total time is O(n).



```java
public void searchProducts(String modelName) {
    HybridList<Furniture> found = new HybridList<>();

    int numOfBranches = system.getNumOfBranches();
    int numOfGoods;
    Branch currentBranch;
    Furniture currentFurniture;

    for(int i = 0; i < numOfBranches; ++i) {
        currentBranch = system.getBranch(i);
        numOfGoods = currentBranch.getNumberOfGoods();

        for(int j = 0; j < numOfGoods; ++j) {
            currentFurniture = currentBranch.getFurniture(j);
            if(currentFurniture.getName().contains(modelName)){
                found.add(currentFurniture);
            }
        }
    }
    seeProducts(found);
}
```

Getting branch from index is O(n). Getting furniture from index is O(n). Adding furniture is amortized constant time. The for loop below is O(n^2). The outer for loop is θ(n). Inside the loop is O(n) + O(n^2) = O(n^2). Total is O(n^3).