Refactoring
2020

Wouter

June 23, 2020

Contents

Gebaseerd op het gelijknamige boek van Martin Fowler & Kent Beck.
Het boek is best een aanrader om te lezen, vooral om na de studies te gebruiken.
De verandering zijn meestal vrij simpel en worden dan ook in combinatie gebruikt.

1 Notes

1.1 Why refactor?

To save time in the long run. Nobody wants to bugfix spaghetti. Mostly to make the code more readable, or encourage reusability.

1.2 Tests

Test everything, all the time. Catch bugs early, fix them while they are small.

1. Refactor something
2. Run Tests
3. Repeat.

2 Code smells

Basically: when should you refactor.
Best to do it while coding, after adding a new feature, this keeps the refactors small and bug free.
No need to refactor passive code, only refactor actively developing code.

- Mysterious name: Naming should be mundane and clear.
  Fixes:
  - Change Function Declaration
  - Rename Variable
  - Rename Field
- Duplicated Code: Keeping multiple identical code up to date is a pain.
  Fixes:
  - Extract Function
  - Slide Statements
  - Pull Up Method
- Long Function: Shorter is better.
  Fixes:
  - Extract Function
  - Replace Temp with Query
  - Introduce Parameter Object
  - Preserve Whole Object
  - Replace Function with Command
  - Decompose Conditional
  - Replace Conditional with Polymorphism
  - Split Loop

- Long Parameter List: Long parameter lists are often confusing.
  Fixes:
  - Replace Parameter with Query
  - Preserve Whole Object
  - Introduce Parameter Object
  - Remove Flag Argument
  - Combine Functions into Class
- Global Data: Hellspawn.
  Fixes:
  - Encapsulate Variable
- Mutable Data: Changes to data can often lead to unexpected consequences and tricky bugs. Control who changes the data and who uses it.
  Fixes:
  - Encapsulate Variable
  - Split Variable
  - Slide Statements
  - Extract Function
  - Separate Query from Modifier
  - Remove Setting Method
  - Replace Derived Variable with Query
  - Combine Functions into Class
  - Combine Functions into Transform
  - Change Reference to Value
- Divergent Change: When we make a change, we want to be able to jump to a single clear point in the system and make the change.
  Fixes:
  - Split Phase
  - Move Function
  - Extract Function
  - Extract Class
- Shotgun Surgery: Opposite of Divergent Change. When making a change and you have to make lots of tiny edits all over the place.
  Fixes:
  - Move Function
  - Move Field
  - Combine Functions into Class
  - Combine Functions into Transform
  - Split Phase
  - Inline Function
  - Inline Class

- Feature Envy: When a function in one module spends more time communicating with functions or data inside another module than it does within its own module.
  Fixes:
  - Move Function
  - Extract Function
- Data Clumps: Groups of data items that pop up together all the time.
  Fixes:
  - Extract Class
  - Introduce Parameter Object
  - Preserve Whole Object
- Primitive Obsession: Avoiding making your own types by butchering primitives.
  Fixes:
  - Replace Primitive with Object
  - Replace Type Code with Subclasses
  - Replace Conditional with Polymorphism
  - Extract Class
  - Introduce Parameter Object
- Repeated Switches: The same conditional switching logic pops up in different places.
  Fixes:
  - Replace Conditional with Polymorphism
- Loops: Pipeline operations help see the elements that are included in the processing and what is done with them.
  Fixes:
  - Replace Loop with Pipeline
- Lazy Element: Removing unneeded structures.
  Fixes:
  - Inline Function
  - Inline Class
  - Collapse Hierarchy
- Speculative Generality: Supporting dreams. "We'll add this ability one day."
  Fixes:
  - Collapse Hierarchy
  - Inline Function
  - Inline Class
  - Change Function Declaration
  - Remove Dead Code
- Temporary Field: A field used only in certain circumstances.
  Fixes:
  - Extract Class
  - Move Function

  - Introduce Special Case

- Message Chains: An object asks an object asks an object . . . for another object.
  Fixes:
  - Hide Delegate
  - Extract Function
  - Move Function

- Middle Man: Too many methods in an interface delegate to another class.
  Fixes:
  - Remove Middle Man
  - Inline Function
  - Replace Superclass with Delegate
  - Replace Subclass with Delegate

- Insider Trading: Too much trading of data.
  Fixes:
  - Move Function
  - Move Field
  - Hide Delegate
  - Replace Subclass with Delegate
  - Replace Superclass with Delegate

- Large Class: A class that wants to do too much.
  Fixes:
  - Extract Class
  - Extract Superclass
  - Replace Type Code with Subclasses

- Alternative Classes with Different Interfaces: Substitution only works if the interfaces are the same.
  Fixes:
  - Change Function Declaration
  - Move Function
  - Extract Superclass

- Data Class: Classes with fields, getters and setters and nothing else.
  Fixes:
  - Encapsulate Record
  - Remove Setting Method
  - Move Function
  - Extract Function
  - Split Phase

- Refused Bequest: Unneeded methods or data from a parent in a child.
  Fixes:
  - Push Down Method
  - Push Down Field

  - Replace Subclass with Delegate
  - Replace Superclass with Delegate

- Comments: Comments shouldn't mask bad code.
  Fixes:
  - Extract Function
  - Change Function Declaration
  - Introduce Assertion

3 A First Set of Refactorings

3.1 Extract Function

Inverse of Inline Function

```
Function printOwing (invoice){
    printBanner();
    let outstanding = calculateOutstanding();
    //print details
    Console.log('name:invoice.customer');
    Console.log('amount:outstanding');
}
```

```
Function printOwing (invoice){
    printBanner();
    let outstanding = calculateOutstanding();
    printDetails(outstanding){
        Console.log('name:invoice.customer');
        Console.log('amount:outstanding');
    }
}
```

3.2 Inline Function

Inverse of Extract Function

```
Function getRating(driver){
    return moreThanFiveLateDeliveries(driver) ? 2 :1;
}
Function moreThanFiveLateDelivers(driver){
    return driver.numberOfLateDelivers > 5;
}
```

```
Function getRating(driver){
    return (driver.numberOfLateDelivers > 5) ? 2 : 1;
}
```

3.3 Extract Variable

Inverse of Inline Variable

```
return order.quantity*order.itemPrice
- Math.max(0,order.quantity-500)*order.itemPrice*0.05
+ Math.min(order.quantity*order.itemPrice*0.1,100);
```

```
const basePrice = order.quantity * order.itemPrice;
const quantityDiscount = Math.max(0,order.quantity - 500) * order.itemPrice * 0.05;
const shipping = Math.min(order.quantity * order.itemPrice * 0.1,100);
return basePrice - quantityDiscount + shipping;
```

3.4 Inline Variable

Inverse of Extract Variable

```
let basePrice = anOrder.basePrice;
return (basePrice > 100);
```

```
return (anOrder.basePrice > 100);
```

3.5 Change Function Declaration

Rename Method, Add Parameter, Remove Parameter

```
Function circum(radius){}
```

```
Function circumference(radius){}
```

3.6 Encapsulate Variable

```
let defaultOwner = {firstname:"Martin",lastname:"Fowler"};
```

```
let defaultOwnerData = {firstname:"Martin",lastname:"Fowler"};
export function defaultOwner(){return defaultOwnerData;}
export setDefaultOwner(arg){defaultOwnerData = arg};
```

3.7 Rename Variable

```
let a = height * width;
```

```
let area = height * width;
```

3.8 Introduce Parameter Object

```
Function amountInvoiced(startDate;endDate;
Function amountReceived(startDate;endDate;
Function amountOverdue(startDate;endDate;
```

```
Function amountInvoiced(aDateRange;
Function amountReceived(aDateRange;
Function amountOverdue(aDateRange;
```

3.9 Combine Functions into Class

```
Function base(aReading){};
Function taxableCharge(aReading){};
Function calculateBaseCharge(aReading){};
```

```
Class Reading{
    Base(){}
    TaxableCharge(){}
    CalculateBaseCharge(){}
}
```

3.10 Combine Functions into Transform

```
Function base(aReading){}
Function taxableCharge(aReading){}
```

```
Function enrichReading(argReading){
    Const aReading = _.cloneDeep(argReading);
    aReading.baseCharge = base(aReading);
    aReading.taxableCharge = taxableCharge(aReading);
    return aReading;
}
```

3.11 Split Phase

```
Const orderData = orderString.split(/\s+/);
Const productPrice = pricelist[orderData[0].split("-")[1]];
Const orderPrice = parseInt(orderData[1])*productPrice;
```

```
Const orderRecord = parseOrder(order);
Const orderPrice = price(orderRecord, priceList);
Function parseOrder(aString){
    Const values = aString.split(/\s+/);
    return({
        productID:values[0].split("-")[1],
        quantity:parseInt(values[1]),
    });
}
Function price (order, pricelist){
    return order.quantity*pricelist[order.productID];
}
```

4 Encapsulation

4.1 Encapsulate Record

Replace record with data class.

```
Organization = {name:"Acme Gooseberries",country:"GB"}
```

```
Class Organization{
    Constructor(data){
        this.name = data.name;
        this.country = data.country;
    }
    get name(){return this.name;}
    set name(arg){this.name = arg;}
    get country(){return this.country;}
    set country(arg){this.country = arg;}
}
```

4.2 Encapsulate Collection

```
Class Person{
    get courses(){return this.courses;}
    set courses(aList){this.courses = aList;}
}
```

```
Class Person{
    get courses(){return this.courses.slice();}
    addCourse(aCourse){}
    removeCourse(aCourse){}
}
```

4.3 Replace Primitive with Object

Replace data value with object, replace type code with class.

```
Orders.filter(o => "high" == o.priority || "rush" == o.priority);
```

↓

```
Orders.filter(o => o.priority.higherThan(new Priority("normal")))
```

4.4 Replace Temp with Query

```
Const basePrice = this.quantity * this.itemPrice;
if(basePrice>1000)
    return basePrice * 0.95;
else
    return basePrice * 0.98;
```

↓

```
get basePrice()this.quantity*this.itemParice;;
...
if(this.basePrice>1000)
    return this.basePrice*0.95;
else
    return this.basePrice*0.98;
```

4.5 Extract Class

Inverse of Inline Class

```
Class Person{
    get officeAreaCode(){return this.officeAreaCode;}
    get officeNumber(){return this.officeNumber;}
}
```

↓

```
Class Person{
    get officeAreaCode(){return this.telephoneNumber.areaCode;}
    get officeNumber(){return this.telephoneNumber.number;}
}
Class TelephoneNumber{
    get areaCode(){return this.areaCode;}
    get number(){return this.number;}
}
```

4.6 Inline Class

Inverse of Extract Class

```
Class Person{
    get officeAreaCode(){return this.telephoneNumber.areaCode;}
    get officeNumber(){return this.telephoneNumber.number;}
}
Class TelephoneNumber{
    get areaCode(){return this.areaCode;}
    get number(){return this.number;}
}
```

```
Class Person{
    get officeAreaCode(){return this.officeAreaCode;}
    get officeNumber(){return this.officeNumber;}
}
```

4.7 Hide Delegate

Inverse of Remove Middle Man

```
manager = aPerson.department.manager;
```

```
manager = aPerson.manager;
Class Person{
    get manager(){return this.department.manager;}
}
```

4.8 Remove Middle Man

Inverse of Hide Delegate.

```
manager = aPerson.manager;
Class Person{
    get manager(){return this.department.manager;}
}
```

```
manager = aPerson.department.manager;
```

4.9 Substitute Algorithm

```
function foundPerson(people){
    for(let i = 0;i < people.length;i++){
        if(people[i] == "Don")
            return "Don";
        if(people[i] == "John")
```

```
            return"Johan";
        if(people[i] == "Kent")
            return "Kent";
    }
    return "";
}
```

↓

```
function foundPerson(people){
    const candidates = ["Don","John","Kent"];
    return people.find(p => candidates.includes(p)) || '';
}
```

5 Moving Features

5.1 Move Function

```
Class Account{
    get overdraftCharge(){}
}
```

↓

```
Class AccountType{
    get overdraftCharge(){}
}
```

5.2 Move Field

```
Class Customer{
    get plan(){return this.plan;}
    get discountRate(){return this.discountRate;}
}
```

↓

```
Class Customer{
    get plan(){return this.plan;}
    get discountRate(){return this.plan.discountRate;}
}
```

5.3 Move Statements into Function

Inverse of Move Statements to Callers

```
Result.push(<p>title:${person.photo.title}</P>);
Result.concat(photoData(person.photo));
Function photoData(aPhoto){
    return[
```

```
        <p>location:${aPhoto.location}</p>,
        <p>date:${aPhoto.date.toDateString()}</p>,
    ];
}
```

↓

```
Result.concat(photoData(person.photo));
Function photoData(aPhoto){
    return[
        <p>title: ${person.photo.title}</P>,
        <p>location:${aPhoto.location}</p>,
        <p>date:${aPhoto.date.toDateString()}</p>,
    ];
}
```

5.4 Move Statements to Callers

Inverse of Move Statements into Function

```
emitPhotoData(outStream,person.photo);
function emitPhotoData(outStream,photo){
    outStream.write(<p>title:{photo.title}</p>\n);
    outStream.write(<p>location:{photo.location}</p>\n);
}
```

↓

```
emitPhotoData(outStream,person.photo);
outStream.write(<p>location:{photo.location}</p>\n);
function emitPhotoData(outStream,photo){
    outStream.write(<p>title:{photo.title}</p>\n);
}
```

5.5 Replace Inline Code with Function Call

```
let appliesToMass = false;
for(const s of states){
    if(s === "MA") appliesToMass = true;
}
```

↓

```
appliesToMass = states.includes("MA");
```

5.6 Slide Statements

```
Const pricingPlan = retrievePricingPlan();
Const order = retreiveOrder();
Let charge;
Const chargePerUnit = pricingPlan.unit;
```

↓

```
Const pricingPlan = retrievePricingPlan();
Const chargePerUnit = pricingPlan.unit;
Const order = retreiveOrder();
Let charge;
```

5.7 Split Loop

```
let averageAge = 0;
let totalSalary = 0;
for(const p of people){
    averageage += p.age;
    totalSalary += p.salary;
}
averageAge = averageAge / people.length;
```

↓

```
let totalSalary = 0;
for(const p of people){
    totalSalary += p.salary;
}
let averageAge = 0;
for(const p of people){
    averageage += p.age;
}
averageAge = averageAge / people.length;
```

5.8 Replace Loop with Pipeline

```
const names = [];
for(const i of input){
    if(i.job === "programmer")
        names.push(i.name);
}
```

↓

```
const names = input
    .filter(i => i.job === "programmer")
    .map(i => i.name)
;
```

5.9 Remove Dead Code

Vrij duidelijk denk ik

6 Organizing Data

6.1 Split Variable

Remove Assignments to Parameters, Split Temp

```
let temp = 2 * (height + width);
console.log(temp);
temp = height * width;
console.log(temp);
```

```
const perimeter = 2 * (height + width);
console.log(perimeter);
const area = height * width;
console.log(area);
```

6.2 Rename Field

```
Class Orgnization{
    get name(){}
}
```

```
Class Orgnization{
    get title(){}
}
```

6.3 Replace Derived Variable with Query

```
get discountedTotal(){return this.discountedTotal;}
set discount(aNumber){
    const old = this.discount;
    this.discount = aNumber;
    this.discountedTotal += old - aNumber;
}
```

```
get discountedTotal(){return this.baseTotal - this.discount;}
set discount(aNumber){this.discount = aNumber;}
```

6.4 Change Reference to Value

Inverse of Change Value to Reference

```
Class Product{
    applyDiscount(arg){this.price.amount -= arg;}
}
```

↓

```
Class Product{
    applyDiscount(arg){
        this.price.amount -= new Money(this.price.amount - arg, this.price.currency);
    }
}
```

6.5 Change Value to Reference

Inverse of Change Reference to Value.

```
let customer = new Customer(customerData);
```

↓

```
let customer = customerRepository.get(customerData.id);
```

7 Simplifying Conditional Logic

7.1 Decompose Conditional

```
if (!aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd))
    charge = quantity * plan.summerRate;
else
    charge = quantity + plan.regularRate + plan.regularServiceCharge;
```

↓

```
if(summer)
    charge = summerCharge();
else
    charge = regularCharge();
```

7.2 Consolidate Conditional Expression

```
if(anEmployee.seniority < 2) return 0;
if(anEmployee.monthsDisabled > 12) return 0;
if(anEmployee.isPartTime) return 0;
```

↓

```
If(isNotEligableForDisability()) return 0;
function isNotEligableForDisability(){
    return ((anEmployee.seniority < 2)
        || anEmployee.monthsDisabled > 12)
        || anEmployee.isPartTime));
}
```

7.3 Replace Nested Conditional with Guard Clauses

```
function getPayAmount(){
    let result;
    if(isDead)
        result = deadAmount();
    else{
        if(isSeparated)
            result = separatedAmount();
        else{
            if(isRetired)
                result = retiredAmount();
            else
                result = normalPayAmount();
        }
    }
    return result;
}
```

```
function getPayAmount(){
    if(isDead) return deadAmount();
    if(isSeparated) return separatedAmount();
    if(isRetired) return retiredAmount();
    return normalPayAmount();
}
```

7.4 Replace Conditional with Polymorphism

```
switch (bird.type){
    case 'EuropeanSwallow':
        return "average";
    case 'AfricanSwallow':
        return(bird.numberOfCocounts > 2) ? "tired" : "average";
    case 'NorwegianBlueParrot':
        return(bird.voltage > 100) ? "scorched" : "beautiful";
    default:
        return "unknown";
}
```

```
class EuropeanSwallow{
    get plumage(){
        return "average";
    }
}
class AfricanSwallow{
    get plumage(){
        return(bird.numberOfCocounts > 2) ? "tired" : "average";
    }
```

```
}
class NorwegianBlueParrot{
    get plumage(){
        return(bird.voltage > 100) ? "scorched" : "beautiful";
    }
}
```

7.5 Introduce Special Case

Introduce Null Object

```
if(aCustomer === "unknown") customerName = "occupant";
```

```
class UnknownCustomer{
    get name(){return "occupant";}
}
```

7.6 Introduce Assertion

```
if(this.discountRate)
    base = base - (this.discountRate * base);
```

```
assert (this.discountRate >= 0);
if(this.discountRate)
    base = base - (this.discountRate * base);
```

8 Refactoring APIs

8.1 Separate Query from Modifier

```
function getTotalOutstandingAndSendBill(){
    const result = customer.invoices.reduce((total,each) => each.amount + total,0);
    sendBill();
    return result;
}
```

```
function totalOutstanding(){
    return customer.invoices.reduce((total,each) => each.amount + total,0);
}
function sendBill(){
    emailGateWay.send(formatBill(customer));
}
```

8.2 Parameterize Function

Parameterize Method.

```
function tenPercentRaise(aPerson){
    aPerson.salary = aPerson.salary.multiply(1.1);
}
function fivePercentRaise(aPerson){
    aPerson.salary = aPerson.salary.multiply(1.05);
}
```

```
function raise(aPerson,factor){
    aPerson.salary = aPerson.salary.multiply(1 + factor);
}
```

8.3 Remove Flag Argument

Replace Parameter with Explicit Methods

```
function setDimension(name,value){
    if(name == "height"){
        this.height = value;
        return;
    }
    if(name == "width"){
        this.width = value;
        return;
    }
}
```

```
function setHeight(value){this.height = value;}
function setWidth(value){this.width = value;}
```

8.4 Preserve Whole Object

```
const low = aRoom.daysTempRange.low;
const high = aRoom.daysTempRange.high;
if(aPlan.withinRange(low,high))
```

```
if(aPlan.withinRange(aRoom.daysTempRange))
```

8.5 Replace Parameter with Query

Inverse of Replace Query with Parameter
Replace Parameter with Method

```
availableVacation(anEmployee,anEmployee.grade);
function availableVacation(anEmployee,grade){
    //calculate vacation
}
```



```
availableVacation(anEmployee)
function availableVacation(anEmployee){
    const grade = anEmployee.grade;
    //calculate vacation
}
```

8.6 Replace Query with Parameter

Inverse of Replace Parameter with Query.

```
targetTemperature(aPlan)
function targetTemperature(aPlan){
    currentTemperature = thermostat.currentTemperature;
    //rest of function
}
```



```
targetTemperature(aPlan, thermostat.currentTemperature)
function targetTemperature(aPlan, currentTemperature){
    //rest of function
}
```

8.7 Remove Setting Method

```
class Person{
    get name(){}
    set name(aString){}
}
```



```
class Person{
    get name(){}
}
```

8.8 Replace Constructor with Factory Function

Replace Constructor with Factory Method

```
leadEngineer = new Employee(document.leadEngineer,'E');
```

↓

```
leadEngineer = createEngineer(document.leadEngineer);
```

8.9 Replace Function with Command

Inverse of Replace Command with Function

```
function score(candidate,medicalExam,scoringGuide){
    let result = 0;
    let healthLevel = 0;
    //long body code
}
```

↓

```
class Score{
    constructor(candidate,medicalExam,scoringGuid){
        this.candidate = candidate;
        this.medicalExam = medicalExam;
        this.scoringGuide = scoringGuide;
    }
    execute(){
        this.result = 0;
        this.healthLevel = 0;
        //long body code
    }
}
```

8.10 Replace Command with Function

Inverse of Replace Function with Command

```
class ChargeCalculator{
    constructor(customer,usage){
        this.customer = customer;
        this.usage = usage;
    }
    execute(){
        return this.customer.rate * this.usage;
    }
}
```

↓

```
function charge(customer,usage){
    return customer.rate * usage;
}
```

9 Dealing with Inheritance

9.1 Pull Up Method

Inverse of Push Down Method

```
class Employee{}
class Salesman extends Employee{
    get name(){}
}
class Engineer extends Employee{
    get name(){}
}
```

```
class Employee{
    get name(){}
}
class Salesman extends Employee{}
class Engineer extends Employee{}
```

9.2 Pull Up Field

Inverse of Push Down Field

```
class Employee{}
class Salesman extends Employee{
    private String name;
}
class Engineer extends Employee{
    private String name;
}
```

```
class Employee{
    protected String name;
}
class Salesman extends Employee{}
class Engineer extends Employee{}
```

9.3 Pull Up Constructor Body

```
class Party{}
class Employee extends Party{
    constructor(name,id,monthlyCost){
        super();
        this.id = id;
        this.name = name;
        this.monthlyCost = monthlyCost;
    }
}
```

↓

```
class Party{
    constructor(name){
        this.name=name;
    }
}
class Employee extends Party{
    constructor(name,id,monthlyCost){
        super(name);
        this.id = id;
        this.monthlyCost = monthlyCost;
    }
}
```

9.4 Push Down Method

Inverse of Pull Up Method

```
class Employee{
    get quota{}
}
class Engineer extends Employee{}
class Salesman extends Employee{}
```

↓

```
class Employee{}
class Engineer extends Employee{}
class Salesman extends Employee{
    get quota{}
}
```

9.5 Push Down Field

Inverse of Pull Up Field

```
class Employee{
    private String quota;
}
class Salesman extends Employee{}
class Engineer extends Employee{}
```

↓

```
class Employee{}
class Engineer extends Employee{}
class Salesman extends Employee{
    protected String quota;
}
```

9.6 Replace Type Code with Subclasses

Inverse of Remove Subclass

```
function createEmployee(name,type){
    return new Employee(name,type);
}
```

```
function createEmployee(name,type){
    switch(type){
        case "engineer" : return new Engineer(name);
        case "salesman" : return new Salesman(name);
        case "manager" : return new Manager(name);
    }
}
```

9.7 Remove Subclass

Replace Subclass with Fields
Inverse of Replace Type Code with Subclasses

```
class Person{
    get genderCode(){ return"X";}
}
class Male extends Person{
    get genderCode(){ return "M";}
}
class Female extends Person{
    get genderCode(){ return "F";}
}
```

```
class Person{
    get genderCode() { return this.genderCode; }
}
```

9.8 Extract Superclass

```
class Department{
    get totalAnnualCost(){}
    get name(){}
    get headcount(){}
}
class Employee{
    get annualCost(){}
    get name(){}
    get id(){}
}
```

```
class Party{
    get annualCost(){}
    get name(){}
}
class Department extends Party{
    get annualCost(){}
    get headcount(){}
}
class Employee extends Party{
    get annualCost(){}
    get id(){}
}
```

9.9 Collapse Hierarchy

```
class Employee{}
class Salesman extends Employee{}
```

```
class Employee{...}
```

9.10 Replace Subclass with Delegate

```
class Order{
    get daysToShip(){
        return this.warehouse.daysToShip;
    }
}
class PriorityOrder extends Order{
    get daysToShip(){
        return this.priorityPlan.daysToShip;
    }
}
```

```
class Order{
    get daysToShip(){
        return (this.priorityDelegate)
            ? this.priorityDelegate.daysToShip
            : this.warehouse.daysToShip;
    }
}
class PriorityOrderDelegate{
    get daysToShip(){
        return this.priorityPlan.daysToShip;
    }
}
```

9.11 Replace Superclass with Delegate

Replace Inheritance with Delegation

```
class List{}
class Stack extends List{}
```

```
class Stack {
    constructor(){
        this.storage = new List();
    }
}
class List{}
```