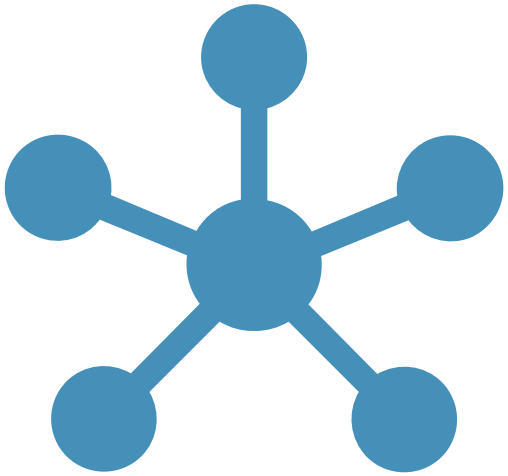
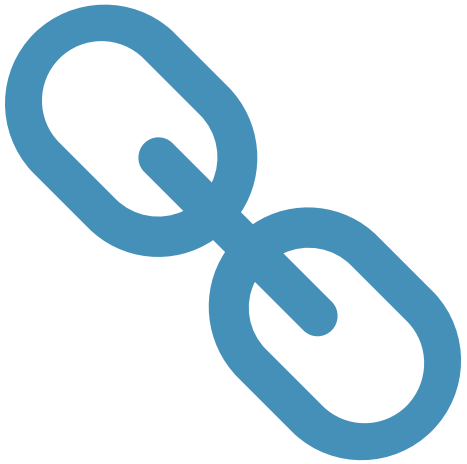
REFACTORING RULES

HOW AND WHEN TO REFACTOR JAVA CODE



TDD / New Development



Maintenance

TWO TIMES WE NEED TO REFACTOR

TDD:ADD SUPPORT FOR A NEW ORDER

```
class ShopTest {

    @Test
    void invoiceHasCorrectDataWhenCustomerBuysProduct() {
        Product product = new Product("Thneed",
                "A thing that everyone needs",
                123.45, 1, 20);
        Shop shop = new Shop();
        Customer customer = new Customer();
        Order order = shop.initiatePurchase(customer, product, 1);
        Invoice invoice = shop.getInvoice(order);
        assertEquals(customer, invoice.customer);
        assertEquals(order, invoice.orders.get(0));
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
class ShopTest {

    @Test
    void invoiceHasCorrectDataWhenCustomerBuysProduct() {
        Product product = new Product("Thneed",
                "A thing that everyone needs",
                123.45, 1, 20);
        Shop shop = new Shop();
        Customer customer = new Customer();
        Order order = shop.initiatePurchase(customer, product, 1);
        Invoice invoice = shop.getInvoice(order);
        assertEquals(customer, invoice.customer);
        assertEquals(order, invoice.orders.get(0));
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
class ShopTest {

    @Test
    void invoiceHasCorrectDataWhenCustomerBuysProduct() {
        Product product = new Product("Thneed",
                "A thing that everyone needs",
                123.45, 1, 20);
        Shop shop = new Shop();
        Customer customer = new Customer();
        Order order = shop.initiatePurchase(customer, product, 1);
        Invoice invoice = shop.getInvoice(order);
        assertEquals(customer, invoice.customer);
        assertEquals(order, invoice.orders.get(0));
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Shop {
    List<Customer> customers;
    List<Customer> vendor;
    List<Invoice> invoices;
}
```

```
public class Customer {
    String name;
    String address;
    String phone;
    String email;
    List<Order> orders;
}
```

```
public class Product {
    String name;
    String description;
    double price;
    double weight;
    int amtInStock;
}
```

```
public class Order {
    Product product;
    int quantity;
}
```

```
public class Invoice {
    Customer customer;
    List<Order> orders;
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Shop {
    List<Customer> customers;
    List<Customer> vendor;
    List<Invoice> invoices;
}
```

```
public class Customer {
    String name;
    String address;
    String phone;
    String email;
    List<Order> orders;
}
```

```
public class Product {
    String name;
    String description;
    double price;
    double weight;
    int amtInStock;
}
```

```
public class Order {
    Product product;
    int quantity;
}
```

```
public class Invoice {
    Customer customer;
    List<Order> orders;
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
class ShopTest {

    @Test
    void invoiceHasCorrectDataWhenCustomerBuysProduct() {
        Product product = new Product("Thneed",
                "A thing that everyone needs",
                123.45, 1, 20);
        Shop shop = new Shop();
        Customer customer = new Customer();
        Order order = shop.initiatePurchase(customer, product, 1);
        Invoice invoice = shop.getInvoice(order);
        assertEquals(customer, invoice.customer);
        assertEquals(order, invoice.orders.get(0));
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
class ShopTest {

    @Test
    void invoiceHasCorrectDataWhenCustomerBuysProduct() {
        Product product = new Product("Thneed",
                "A thing that everyone needs",
                123.45, 1, 20);
        Shop shop = new Shop();
        Customer customer = new Customer();
        Order order = shop.initiatePurchase(customer, product, 1);
        Invoice invoice = shop.getInvoice(order);
        assertEquals(customer, invoice.customer);
        assertEquals(order, invoice.orders.get(0));
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```java
public class Product {
    String name;
    String description;
    double price;
    double weight;

    public Product(String name, String description, double price, double weight, int amtInStock) {
        this.name = name;
        this.description = description;
        this.price = price;
        this.weight = weight;
        this.amtInStock = amtInStock;
    }

    int amtInStock;
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
class ShopTest {

    @Test
    void invoiceHasCorrectDataWhenCustomerBuysProduct() {
        Product product = new Product("Thneed",
                "A thing that everyone needs",
                123.45, 1, 20);
        Shop shop = new Shop();
        Customer customer = new Customer();
        Order order = shop.initiatePurchase(customer, product, 1);
        Invoice invoice = shop.getInvoice(order);
        assertEquals(customer, invoice.customer);
        assertEquals(order, invoice.orders.get(0));
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Shop {
    List<Customer> customers;
    List<Customer> vendor;
    Map<Order, Invoice> invoices = new HashMap<>();

    public Order initiatePurchase(Customer customer, Product product, int quantity) {
        Order order = new Order(product, quantity);
        invoices.put(order, new Invoice(customer, new ArrayList<>(Arrays.asList(order))));
        return order;
    }

    public Invoice getInvoice(Order order) {
        return invoices.get(order);
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Shop {
    List<Customer> customers;
    List<Customer> vendor;
    Map<Order, Invoice> invoices = new HashMap<>();

    public Order initiatePurchase(Customer customer, Product product, int quantity) {
        Order order = new Order(product, quantity);
        invoices.put(order, new Invoice(customer, new ArrayList<>(Arrays.asList(order))));
        return order;
    }

    public Invoice getInvoice(Order order) {
        return invoices.get(order);
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Order {
    Product product;
    int quantity;

    public Order(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Shop {
    List<Customer> customers;
    List<Customer> vendor;
    Map<Order, Invoice> invoices = new HashMap<>();

    public Order initiatePurchase(Customer customer, Product product, int quantity) {
        Order order = new Order(product, quantity);
        invoices.put(order, new Invoice(customer, new ArrayList<>(Arrays.asList(order))));
        return order;
    }

    public Invoice getInvoice(Order order) {
        return invoices.get(order);
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Invoice {
    Customer customer;
    List<Order> orders;

    public Invoice(Customer customer, List<Order> orders) {
        this.customer = customer;
        this.orders = orders;
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
class ShopTest {

    @Test
    void invoiceHasCorrectDataWhenCustomerBuysProduct() {
        Product product = new Product("Thneed",
                "A thing that everyone needs",
                123.45, 1, 20);
        Shop shop = new Shop();
        Customer customer = new Customer();
        Order order = shop.initiatePurchase(customer, product, 1);
        Invoice invoice = shop.getInvoice(order);
        assertEquals(customer, invoice.customer);
        assertEquals(order, invoice.orders.get(0));
    }
}
```

TDD: ADD SUPPORT FOR A NEW ORDER

```
class ShopTest {

    @Test
    void invoiceHasCorrectDataWhenCustomerBuysProduct() {
        Product product = new Product("Thneed",
                "A thing that everyone needs",
                123.45, 1, 20);
        Shop shop = new Shop();
        Customer customer = new Customer();
        Order order = shop.initiatePurchase(customer, product, 1);
        Invoice invoice = shop.getInvoice(order);
        assertEquals(customer, invoice.customer);
        assertEquals(order, invoice.orders.get(0));
    }
}
```

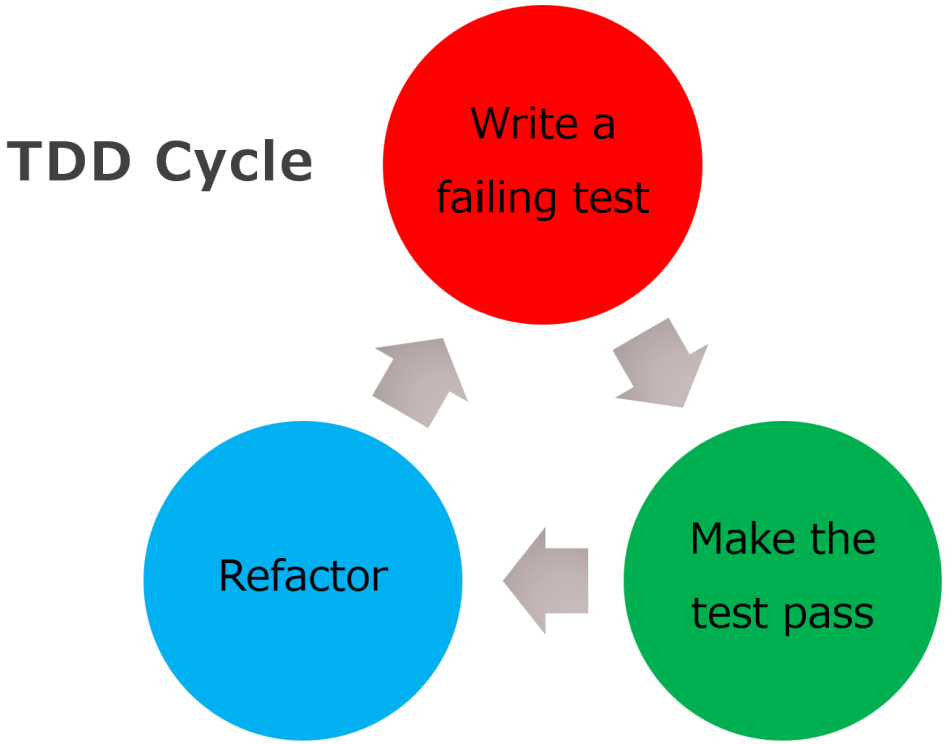
TDD: ADD SUPPORT FOR A NEW ORDER

It works! Move on!

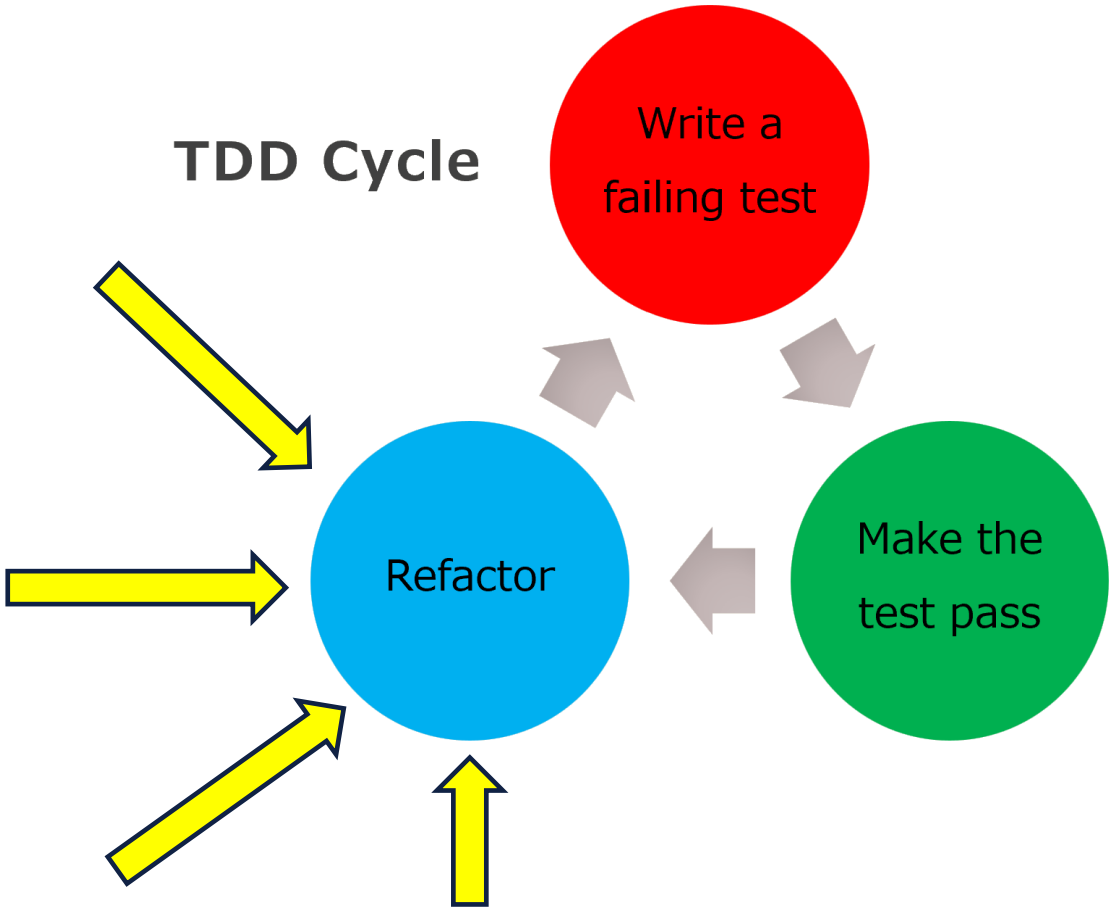
TDD: ADD SUPPORT FOR A NEW ORDER

~~It works! Move on!~~

TDD:ADD SUPPORT FOR A NEW ORDER



TDD:ADD SUPPORT FOR A NEW ORDER



TDD:ADD SUPPORT FOR A NEW ORDER

```java
public class Product {
    String name;
    String description;
    double price;
    double weight;

    public Product(String name, String description, double price, double weight, int amtInStock) {
        this.name = name;
        this.description = description;
        this.price = price;
        this.weight = weight;
        this.amtInStock = amtInStock;
    }

    int amtInStock;
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
1 usage   scott
public Product(String name, String description, double price, double we
    this.na
    this.de
    this.pr
    this.we
    this.am
}

2 usages
```

- Change access modifier
- Generate missed test methods
- Insert '@Contract(pure = true)'
- Replace constructor with builder
- Replace constructor with factory method
- Add Javadoc
- Generate overloaded constructor with default parameter values

Press Ctrl+Q to toggle preview

TDD:ADD SUPPORT FOR A NEW ORDER

```java
public class Product {
    String name;
    String description;
    double price;
    double weight;

    public Product(String name, String description, double price, double weight, int amtInStock) {
        this.name = name;
        this.description = description;
        this.price = price;
        this.weight = weight;
        this.amtInStock = amtInStock;
    }

    int amtInStock;
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Shop {
    List<Customer> customers;
    List<Customer> vendor;
    Map<Order, Invoice> invoices = new HashMap<>();

    public Order initiatePurchase(Customer customer, Product product, int quantity) {
        Order order = new Order(product, quantity);
        invoices.put(order, new Invoice(customer, new ArrayList<>(Arrays.asList(order))));
        return order;
    }

    public Invoice getInvoice(Order order) {
        return invoices.get(order);
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Shop {
    List<Customer> customers;
    List<Customer> vendor;
    Map<Order, Invoice> invoices = new HashMap<>();

    public Order initiatePurchase(Customer customer, Product product, int quantity) {
        Order order = new Order(product, quantity);
        invoices.put(order, new Invoice(customer, new ArrayList<>(Arrays.asList(order))));
        return order;
    }

    public Invoice getInvoice(Order order) {
        return invoices.get(order);
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Shop {
    List<Customer> customers;
    List<Customer> vendor;
    Map<Order, Invoice> invoices = new HashMap<>();

    public Order initiatePurchase(Customer customer, Product product, int quantity) {
        Order order = new Order(product, quantity);
        invoices.put(order, new Invoice(customer, new ArrayList<>(Arrays.asList(order))));
        return order;
    }

    public Invoice getInvoice(Order order) {
        return invoices.get(order);
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Invoice {
    Customer customer;
    List<Order> orders;

    public Invoice(Customer customer, List<Order> orders) {
        this.customer = customer;
        this.orders = orders;
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Order {
    Product product;
    int quantity;

    public Order(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
class ShopTest {

    @Test
    void invoiceHasCorrectDataWhenCustomerBuysProduct() {
        Product product = new Product("Thneed",
                "A thing that everyone needs",
                123.45, 1, 20);
        Shop shop = new Shop();
        Customer customer = new Customer();
        Order order = shop.initiatePurchase(customer, product, 1);
        Invoice invoice = shop.getInvoice(order);
        assertEquals(customer, invoice.customer);
        assertEquals(order, invoice.orders.get(0));
    }
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Customer {
    String name;
    String address;
    String phone;
    String email;
    List<Order> orders;
}
```

TDD: ADD SUPPORT FOR A NEW ORDER

```
@Test
void invoiceHasCorrectDataWhenCustomerBuysProduct() {
    Product product = new Product("Thneed",
            "A thing that everyone needs",
            123.45, 1, 20);
    Shop shop = new Shop();
    Customer customer = new Customer("Mary Lou Who",
            "123 Main St., Whoville", "555-1212", "marylou@who.com");
    Order order = shop.initiatePurchase(customer, product, 1);
    Invoice invoice = shop.getInvoice(order);
    assertEquals(customer, invoice.customer);
    assertEquals(order, invoice.orders.get(0));
}
```

TDD:ADD SUPPORT FOR A NEW ORDER

```
public class Customer {
    String name;
    String address;
    String phone;
    String email;
    List<Order> orders;

    public Customer(String name, String address, String phone, String email) {
        this.name = name;
        this.address = address;
        this.phone = phone;
        this.email = email;
    }
}
```

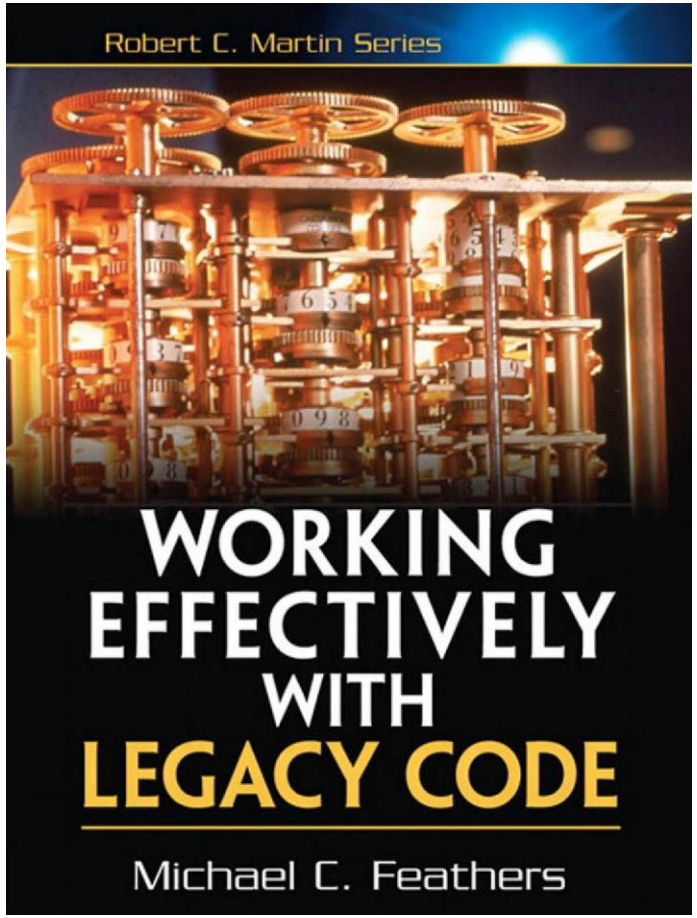
TDD: ADD SUPPORT FOR A NEW ORDER

```
@Test
void invoiceHasCorrectDataWhenCustomerBuysProduct() {
    Product product = new Product("Thneed",
            "A thing that everyone needs",
            123.45, 1, 20);
    Shop shop = new Shop();
    Customer customer = new Customer("Mary Lou Who",
            "123 Main St., Whoville", "555-1212", "marylou@who.com");
    Order order = shop.initiatePurchase(customer, product, 1);
    Invoice invoice = shop.getInvoice(order);
    assertEquals(customer, invoice.customer);
    assertEquals(order, invoice.orders.get(0));
}
```

TDD: ADD SUPPORT FOR TWO NEW ORDERS

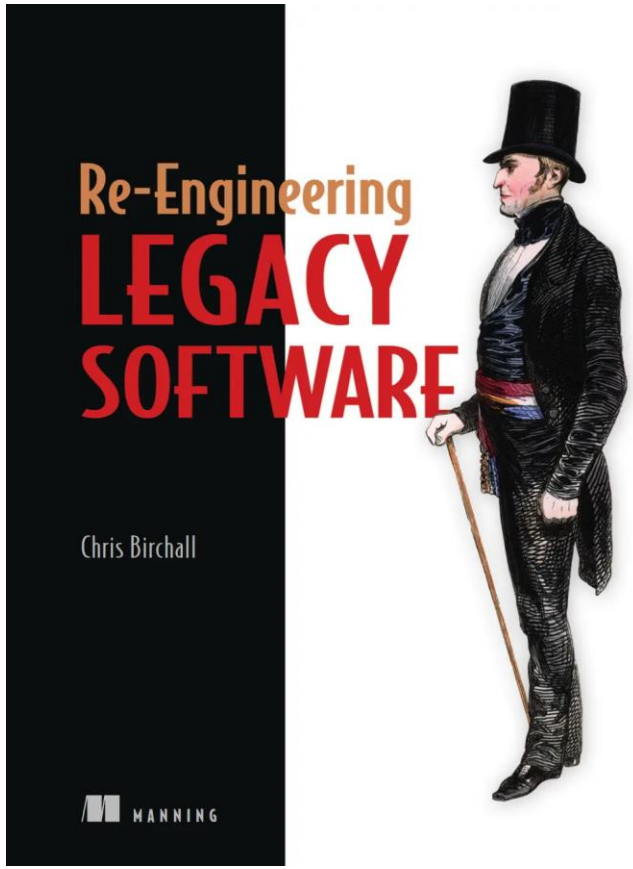
```
@Test
void invoiceReportsTwoProductsForOneCustomerWhenCustomerBuysTwoProducts() {
    Product thneed = new Product("Thneed",
            "A thing that everyone needs",
            123.45, 1, 20);
    Product hinkleHorn = new Product("Hinkle-horn",
            "Includes hooks upon which it can be hung while sleeping",
            87.65, 1, 2);
    Shop shop = new Shop();
    Customer customer = new Customer();
    Order order = shop.initiatePurchase(customer, thneed, 1);
    Order order2 = shop.initiatePurchase(customer, hinkleHorn, 1);
    Invoice invoice = shop.getInvoice(order);
    assertEquals(customer, invoice.customer);
    assertEquals(order, invoice.orders.get(0));
    assertEquals(order2, invoice.orders.get(1));
}
```

REFACTORING EXISTING CODE



Excellent resource!

WHOLE APPROACH TO LEGACY APPLICATIONS



Another great one!

BOY SCOUTS RULE OF CAMPING

Always leave the campsite cleaner than you found it.

BOY SCOUTS RULE OF CAMPING

"Clean as you code"

- Olivier Gaudin, SonarSource

REMOVE COMMENTED-OUT CODE

```
{
  font = new Font("Bookman Old Style", Font.BOLD, 30);
  g.setFont(font);
  g.setColor(Color.WHITE);
  //          if (otherData ≠ "")
  //          {
  //              if (otherData == "")
  //              {
  //                  otherData = GetDefaultData();
  //                  StringBuilder x = new StringBuilder(50000);
  //                  for (int i = 0; i < 20; i++)
  g.drawString(data3point14[0], 145, 205);
  //                  {
  //                      x.Append(char.ToUpper(otherData[i]));
  //                  }
  //              }
  //              boundingRect = new RectangleF(50, 100, 320, 320);
  //              g.DrawString(otherData, new Font("Cooper Black", 40), new SolidBrush(Color.White),
  //          }
  g.drawString(data3point14[1], 170, 235);
}
```

REMOVE UNNECESSARY COMMENTS

```
/**
 *
 */
private String[] horizontalLabelNames;
/**
 * It's the vertical label names
 */
private String[] verticalLabelNames;
/**
 * John says that this is better than the old way
 */
private int      ct;
```

REMOVE OUTDATED AND UNNECESSARY COMMENTS

```java
/**
 * Shows the chart
 *
 * @param ct
 * @param jjReq1205
 * @param orientation
 * @param reversornotreverse
 * @param jackshiddenhack
 * @return
 */
public void iniDS(int ct, String stjjDReq1205, boolean b)
{
  this.ct = ct;
  this.jjD = stjjDReq1205;
  // Changed by Sally 2/14
  if (b)
  {
    iHATEthisUckingJob();
  }
}
```

REMOVE UNHELPFUL COMMENTS

```
/**
 *
 * @return
 */
private Unit horizontalNaming()
{
  return new Unit();
@Override
public Set<AWTKeyStroke> getFocusTraversalKeys(int id)
{
  // TODO Auto-generated method stub
  return super.getFocusTraversalKeys(id);
}
```

REMOVE DEAD CODE

```
public boolean isPlayable() {
    return (howMany
}

3 usages   Sébastien Nico
public boolean add(
```

Method 'isPlayable()' is never used

Safe delete 'isPlayable()' Alt+Shift+Enter More actions... Alt+Enter

com.adaptionsoft.games.uglytrivia.Game

```
public boolean isPlayable()
```

java

REMOVE USELESS TESTS

```
@Test
public void testFunction() {
    ConfirmationLetterGenerator generator = new ConfirmationLetterGenerator();
    generator.setCurrencyDao(new CurrencyDao());
}
```

REMOVE FRAGILE / FINE-GRAINED TESTS

```
@Test
public void testProperShopInitialization() {
    Shop shop = new Shop();
    shop.addCustomer(new Customer("Bob", "123 Elm St.", "555-1212", "bob@bob.com"));
    shop.addVendor(new Customer("Sears", "123 Sears Ave.", "867-5309", "solid@sears.com));
    assertEquals(shop.getCustomer(1).getName().equals("Bob"));
    assertEquals(shop.getVendor(1).getName().equals("Sears"));
}
```

CODEGLANCE PRO

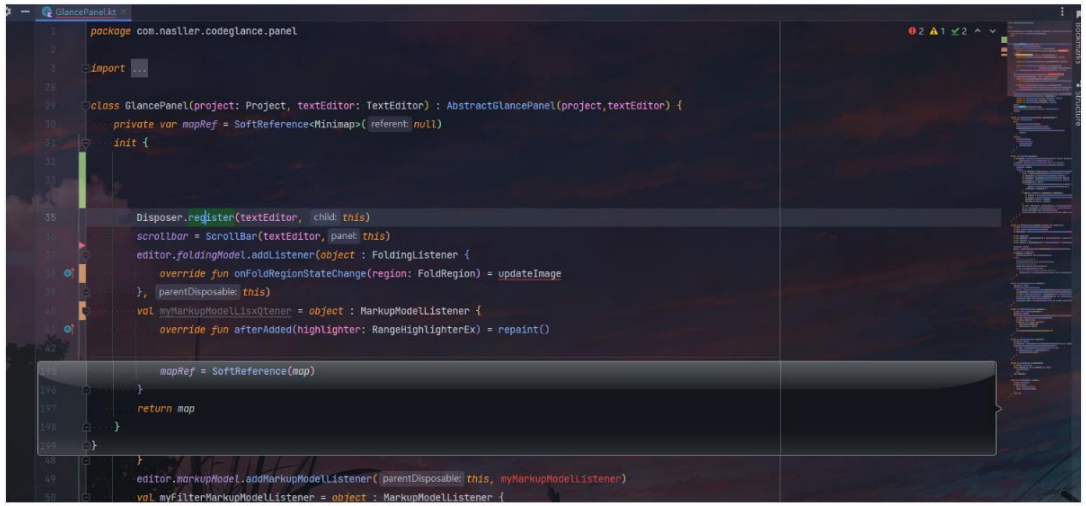
Code Editing Viewer

CodeGlance Pro

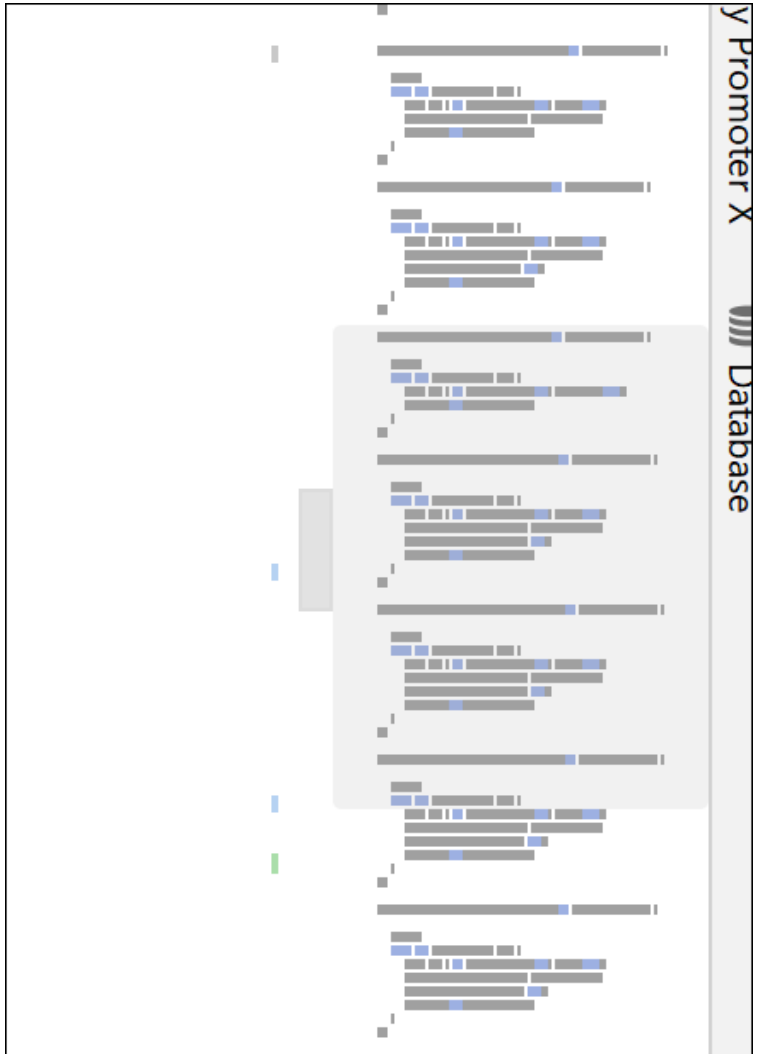
★ ★ ★ ★ ★

Nasller

Overview Versions Reviews



DUPLICATED CODE



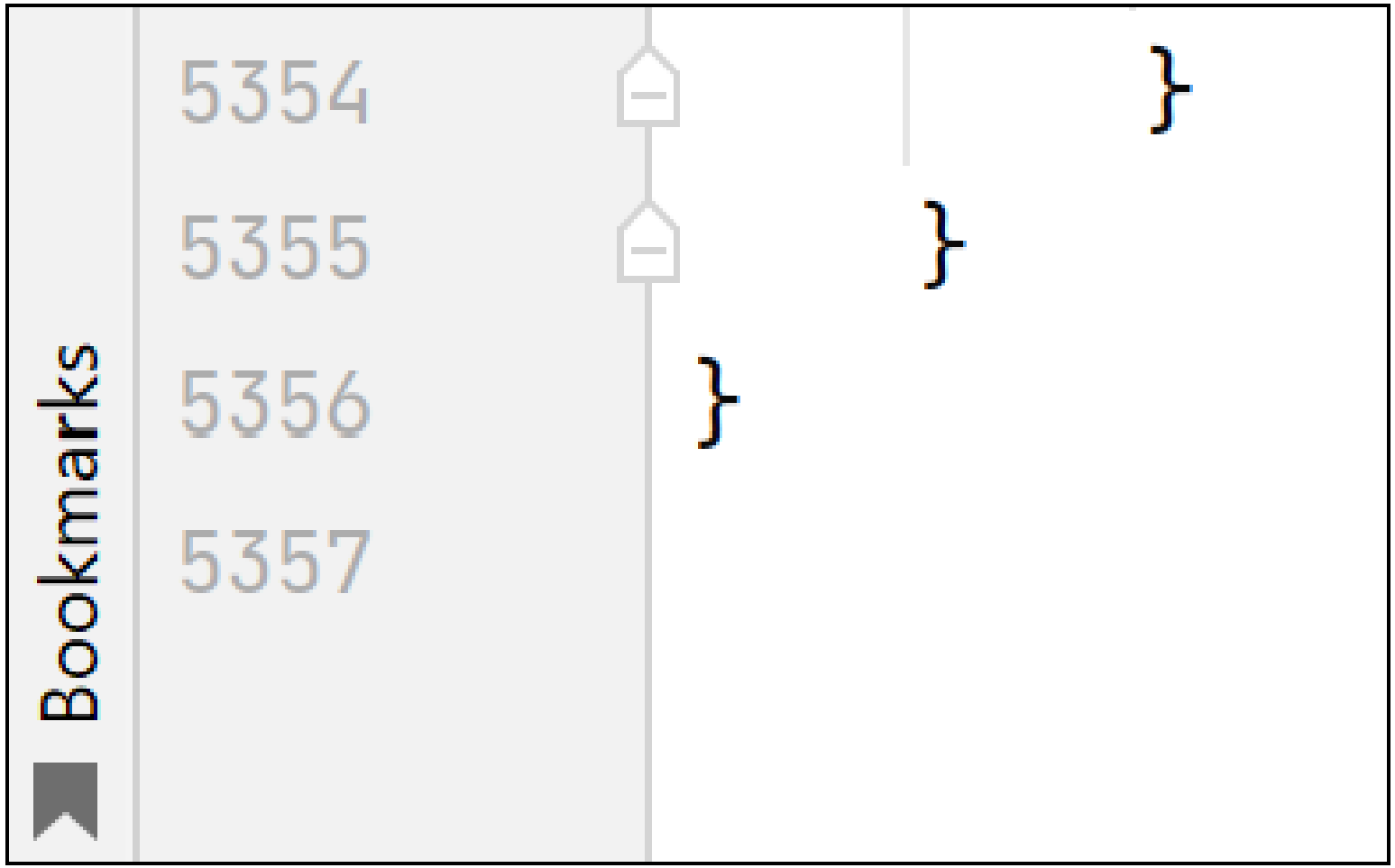
DUPLICATED CODE

```
header.findViewById(R.id.history).setOnClickListener(new OnSingleClickListener() {
    nxt3 +1
    @Override
    public void onSingleClick(View view) {
        Intent inte = new Intent(MainActivity.this, Profile.class);
        inte.putExtra(Profile.EXTRA_PROFILE, Authentication.name);
        inte.putExtra(Profile.EXTRA_HISTORY, true);
        MainActivity.this.startActivity(inte);
    }
});
nxt3
header.findViewById(R.id.commented).setOnClickListener(new OnSingleClickListener() {
    nxt3
    @Override
    public void onSingleClick(View view) {
        Intent inte = new Intent(MainActivity.this, Profile.class);
        inte.putExtra(Profile.EXTRA_PROFILE, Authentication.name);
        inte.putExtra(Profile.EXTRA_COMMENT, true);
        MainActivity.this.startActivity(inte);
    }
});
```

DUPLICATED CODE

```
header.findViewById(R.id.history).setOnClickListener(new OnSingleClickListener() {
    nxt3 +1
    @Override
    public void onSingleClick(View view) {
        Intent inte = new Intent(MainActivity.this, Profile.class);
        inte.putExtra(Profile.EXTRA_PROFILE, Authentication.name);
        inte.putExtra(Profile.EXTRA_HISTORY, true);
        MainActivity.this.startActivity(inte);
    }
});
nxt3
header.findViewById(R.id.commented).setOnClickListener(new OnSingleClickListener() {
    nxt3
    @Override
    public void onSingleClick(View view) {
        Intent inte = new Intent(MainActivity.this, Profile.class);
        inte.putExtra(Profile.EXTRA_PROFILE, Authentication.name);
        inte.putExtra(Profile.EXTRA_COMMENT, true);
        MainActivity.this.startActivity(inte);
    }
});
```

LONG CLASS / METHOD



LONG PARAMETER LIST

```
public MyOutStream letter(RequestContext context,
                          FileUploadCommand fileUploadCommand, Client client,
                          HashBatchRecordsBalance hashBatchRecordsBalance, String branchName,
                          List<AmountAndRecordsPerBank> bankMap,
                          List<com.example.record.domain.FaultRecord> faultyRecords,
                          FileExtension extension, List<Record> records,
                          List<TempRecord> faultyAccountNumberRecordList,
                          List<TempRecord> sansDuplicateFaultRecordsList
) {
```

REPLACE WITH A CLASS OBJECT + BUILDER PATTERN

```
public class LetterParamsBuilder {
    private RequestContext context;
    // ...
    public LetterParamsBuilder setContext(RequestContext context) {
        this.context = context;
        return this;
    }
    // ...
    public ConfirmationLetterGenerator.LetterParams createLetterParams() {
        return new ConfirmationLetterGenerator.LetterParams(context, fileUploadCommand,
    }
```

WHERE TO BEGIN

TOOLS (+ CONSULTING) TO HELP

- CodeClimate
- SilverThread
- CodeScene

CODECLIMATE REPORT

Showing 1,550 of 2,925 total issues

File `MainActivity.java` has 4577 lines of code (exceeds 250 allowed). Consider refactoring. OPEN

```
1  package me.ccrama.redditslide.Activities;
2
3  import android.Manifest;
4  import android.animation.Animator;
5  import android.animation.ArgbEvaluator;
```

Found in app/src/main/java/me/ccrama/redditslide/Activities/MainActivity.java About 1 wk to fix

CODECLIMATE REPORT

```
Method `onActivityResult` has a Cognitive Complexity of 43 (exceeds 5 allowed). Consider refactorin
Method `onActivityResult` has 66 lines of code (exceeds 25 allowed). Consider refactoring.
```

```
246     @Override
247     protected void onActivityResult(int requestCode, int resultCode, Intent data) {
248         if (requestCode == SETTINGS_RESULT) {
249             int current = pager.getCurrentItem();
250             if (commentPager && current == currentComment) {
251                 current = current - 1;
252             }
253             if (current < 0) current = 0;
254             adapter = new OverviewPagerAdapter(getSupportFragmentManager());
255             pager.setAdapter(adapter);
256             pager.setCurrentItem(current);
257             if (mTabLayout != null) {
```

MOST CHANGED FILES IN GIT REPOSITORY

Google: “Git most changed files”

MOST CHANGED FILES IN GIT REPOSITORY

Google: “Git most changed files”

```
git log --pretty=format:"" --name-only | grep "[^\s]" | sort | uniq -c | sort -nr | head -10
```

ADD TESTS TO UNTESTABLE CODE

Approval Tests

Home Builds Platforms Resources Blog F.A.Q.

Home

> *A picture's worth a 1000 tests.*

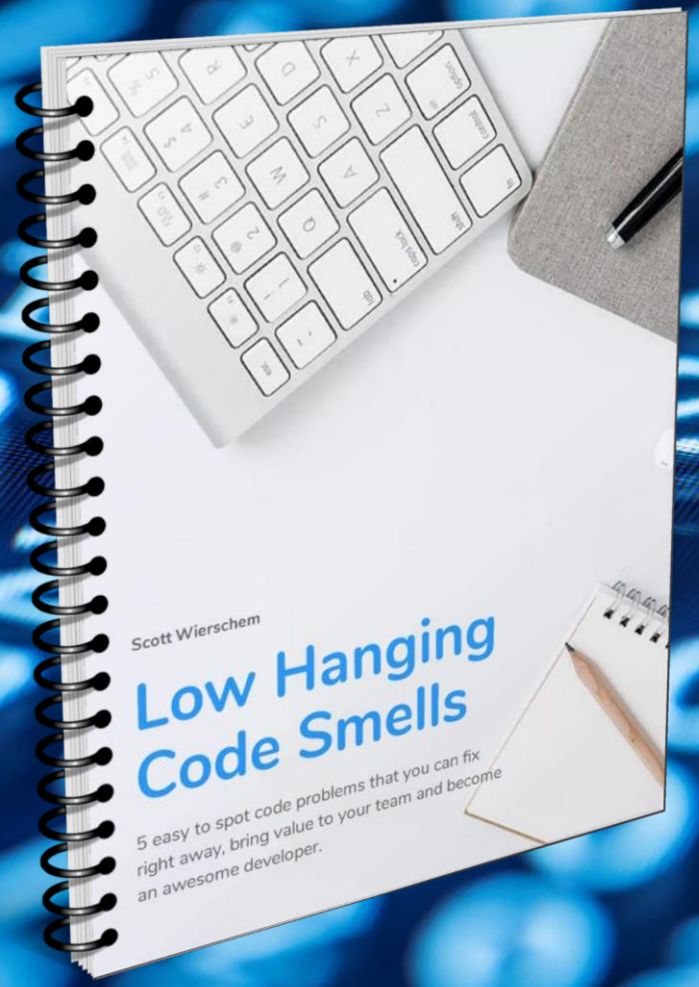
Unit testing asserts can be difficult to use. Approval tests simplify this by taking a snapshot of the results, and confirming that they have not changed.

In normal unit testing, you say `assertEquals(5, person.getAge())`. Approvals allow you to do this when the thing that you want to assert is no longer a primitive but a complex object. For example, you can say, `Approvals.verify(person).`

Note: Approvaltests is in many languages. We suggest you start there as it is the best maintained for each project.

Java python Lua [OBJ-C] Ruby

https://approvaltests.com



https://jav.mn/codesmells

THANK YOU

@SCOTT_WIERSCHEM

KEEPCALMANDREFACTOR.COM