Certified Scrum Developer
What Smells?

Example 1–Code is part of same Class

```
public class InMemoryCustomerRepository implements
CustomerRepository {

private Map<String, Customer> allCustomers;

private void setInitialSeedData() {
Customer customer1 = new Customer();
customer1.setLogin("mmaltiar");
customer1.setFirstName("Meetu");
customer1.setLastName("Maltiar");
customer1.setAddress("Gurgaon");
allCustomers.put("meetum", customer1);

Customer customer2 = new Customer();
customer2.setLogin("vhazrati");
customer2.setFirstName("Vikas");
customer2.setLastName("Hazrati");
customer2.setAddress("New Delhi");
allCustomers.put("vhazrati", customer2);

Customer customer3 = new Customer();
customer3.setLogin("schillara");
customer3.setFirstName("Srinivas");
customer3.setLastName("Chillara");
customer3.setAddress("Pune");
allCustomers.put("schillara", customer3);
}

// Other Methods
}
```

```
public class Customer {

private String login;

private String firstName;

private String lastName;

private String address;

public Customer() {
}

//Getters and Setters

}
```

Example 2–Sibling Classes

```
public class LargeCustomerHandler {

     public void
sendPromotionalMailToFilteredLargeCustomers
(Collection<Customer> allPremiumCustomers) {
  Collection<Customer> filteredCustomers =
filterCustomersForPromotion(allPremiumCustomers);
  for (Customer customer : filteredCustomers) {
  sendMail(customer);
  }
  }

      private Collection<Customer>
filterCustomersForPromotion
(Collection<Customer> allCustomers)  {
  // Actual filtering Logic
  return null;
  }

  private void sendMail(Customer customer) {
  //Invoke actual Mail sending program
  }
}
```

```
public class MidSizeCustomerHandler {

public void
sendPromotionalMailToFilteredMidSizeCustomers
(Collection<Customer> allMidSizeCustomers) {
  Collection<Customer> filteredCustomers =
filterCustomersForPromotion(allMidSizeCustomers);
  for (Customer customer : filteredCustomers) {
  sendMail(customer);
  }
  }

  private Collection<Customer>
filterCustomersForPromotion
                    (Collection<Customer>
allCustomers)  {
  // Actual filtering Logic
  return null;
  }

  private void sendMail(Customer customer) {
  //Invoke actual Mail sending program
  }
}
```

Example 3

```
public boolean validateAppliedLeave(Date appliedDate, int numberDays, String leaveType, Employee employee) {
  Calendar appliedDateCalendar = Calendar.getInstance();
  appliedDateCalendar.setTime(appliedDate);
  // Check whether the applied date is not a weekly day-off : Saturday or Sunday
  int dayOfWeek = appliedDateCalendar.get(Calendar.DAY_OF_WEEK);
  if (dayOfWeek == Calendar.SATURDAY || dayOfWeek == Calendar.SUNDAY) {
  return false;
  }
  // Check whether the applied date is not a public holiday
  // some logic

  // Check whether the applied number of days are more than the current remaining quota
  if (leaveType.equals(CASUAL_LEAVE) && numberDays > employee.getRemainingCasualLeaves()) {
  return false;
  } else if (leaveType.equals(PAID_LEAVE) && numberDays > employee.getRemainingPaidLeaves()) {
  return false;
  } else if (leaveType.equals(SICK_LEAVE) && numberDays > employee.getRemainingSickLeaves()) {
  return false;
  }
  // Check whether the applied number of days corresponds to leave-types quota policies
  if (leaveType.equals(CASUAL_LEAVE) && numberDays > MAX_CONSECUTIVE_CASUAL_LEAVES) {
  return false;
  } else if (leaveType.equals(PAID_LEAVE) && numberDays > MAX_CONSECUTIVE_PAID_LEAVES) {
  return false;
  } else if (leaveType.equals(SICK_LEAVE) && numberDays > MAX_CONSECUTIVE_SICK_LEAVES) {
  return false;
  }
  return true;
}
```

Example 4

```
public class EmployeeLeaveApprover {

private final String CASUAL_LEAVE = "casualLeave";
private final int MAX_CONSECUTIVE_CASUAL_LEAVES = 2;
private List<Date> publicHolidays;
// More instance variables
public LeaveApprovalInformation
         validateAndApproveLeaveApplication(Date appliedDate,
           int numberDays, String leaveType, Employee employee)  {
  if (!validateAppliedDateWithWeekendsAndPublicHolidays(appliedDate) &&
  !validateLeaveQuotaOfEmployee(numberDays, leaveType) &&
  !validateNumberOfAppliedDaysWithLeaveTypePolicy(leaveType, numberDays)) {

  //create Appropriate LeaveApproval Information object and return
  LeaveApprovalInformation result = new LeaveApprovalInformation();
  result.setStatus(LeaveApprovalStatus.REJECTED);
  result.setMessage("Your leave request didn't meet the laid out policies");
  }

  //If validation rules pass successfully, send approval requests to relevant Managers
  sendApprovalRequestToEmployeeLineManager(employee);
  sendApprovalRequestToEmployeeProjectManagers(employee);
  //More Logic
}
private boolean validateAppliedDateWithWeekendsAndPublicHolidays(Date appliedDate) {
  // Relevant logic
  return false;
}
private boolean validateNumberOfAppliedDaysWithLeaveTypePolicy(String leaveType, int numberDays) {
  // Relevant logic
  return false;
}
// More validateMethods
private void sendApprovalRequestToEmployeeLineManager(Employee employee) {
// Relevant logic
}
// More sendApprovalMethods
private void sendStatusInformationToRequestedEmployee(Employee employee) {
// Relevant logic
}
```

Example 5

```
public class Person {

   // Personal Information

  private String firstName;
  private String lastName;
  private Date dateOfBirth;
  private Gender gender;

  // Address Information

  private String houseNumber;
  private String streetName;
  private String addressSecondLine;
  private String city;
  private String postalCode;
  private String zipCode;

  // Contact Information

  private String areaCode;
  private String landlinePhoneNumber;
  private String mobilePhoneNumber;
  private String mailId;

//Getters and Setters

//Other methods
```

Example 6

```
public interface AlertProcessor {

  Collection<CompanyAlert> processAlerts(Collection<CompanyAlert> incomingAlerts, AlertType alertType);
}
```



```
public abstract class BaseAlertProcessor implements AlertProcessor{

  protected AlertType alertType;
  protected boolean isProcessorApplicable(AlertType alertType) {
  return this.alertType.equals(alertType);
  }
}
```

```
public class ScoreAlertProcessor extends
BaseAlertProcessor {
         public ScoreAlertProcessor() {
    this.alertType = AlertType.SCORE;
         }
          @Override
          public Collection<CompanyAlert> processAlerts
  (Collection<CompanyAlert> incomingAlerts, AlertType
alertType) {

          Collection<CompanyAlert> result = new
ArrayList<CompanyAlert>();
          if (isProcessorApplicable(alertType)) {
               // Logic here
          }
         return result;
         }
}
```

```
public class NewsAlertProcessor extends
BaseAlertProcessor {
         public ScoreAlertProcessor() {
    this.alertType = AlertType.NEWS;
         }
          @Override
          public Collection<CompanyAlert> processAlerts
  (Collection<CompanyAlert> incomingAlerts, AlertType
alertType) {

          Collection<CompanyAlert> result = new
ArrayList<CompanyAlert>();
          if (isProcessorApplicable(alertType)) {
               // Logic here
          }
         return result;
         }
}
```

Example 7

```
public class Address {

private String houseNumber;
private String firstLine;
private String secondLine;
private String city;
private String postalCode;
private String state;
private String region;
private String postOffice;
private String country;
//More Methods
}
```

Example 8–Client wants to know a department's manager

```
public class Consultant {

  private Department department;

  private String firstName;

  private String lastName;

  private Date joiningDate;

  public Department getDepartment() {
  return department;
  }
  //More methods
}
```

```
public class Department {

private String name;

  private Consultant manager;

  private Set<Consultant> members = new
HashSet<Consultant>();

  //More methods
}
```