

DMCommunity.org

Challenge Mar-2023 “Permit Eligibility”

Analysis of Solution Generated by ChatGPT-4 on April 2, 2023

ChatCPT has already produced a nice-looking [solution](#) even with a pseudo-code. However, anybody who looks at it will see that it was obviously incorrect. In particular, it does not accumulate intersecting periods, does not respect 10-years limit, and makes more errors. Instead of criticizing the current ChatGPT capabilities, I thought I should try to help ChatGPT to produce a more reasonable solution. Below I described the results of my efforts split into 3 parts:

- [Part 1. My Dialog with ChatGPT](#)
- [Part 2. Validation of the ChatGPT Solution](#)
- [PART 3. Manual Conversion of ChatGPT Solution to Working Java Code](#)

PART 1. My dialog with ChatGPT

Request:

Build a decision model for this problem. As part of a regulatory process, a government agency wants to determine if an applicant is eligible for a resident permit using a simple rule: an applicant is eligible for a resident permit if the applicant has lived at an address while married and in that time period, they have shared the same address at least 7 of the last 10 years. Here is an example of input data:

A list of periods living at an address for applicant (From, To, Address):

2010-01-01,2015-12-31,"123 Main St, Anytown, USA"

2016-01-01,2020-12-31,"456 Oak St, Anytown, USA"

2021-01-01,2023-03-04,"789 Elm St, Anytown, USA"

A list of periods living at an address for spouse (From, To, Address):

2010-01-01,2015-12-31,"123 Main St, Anytown, USA"

2016-01-01,2020-12-31,"120 Maple St, Anytown, USA"

2021-01-01,2023-03-04,"789 Elm St, Anytown, USA"

A list of applicant and spouse marriage periods (From, To):

2010-01-01,2015-12-31

2021-01-01,2023-03-04

Response:



To build a decision model for this problem, we can follow the steps given below:

Step 1: Extract the relevant data from the given input, i.e., periods of the applicant and the spouse living at different addresses, and periods of their marriage.

Step 2: For each period in which the applicant lived at an address while being married, check if the spouse also lived at the same address during that period. If yes, mark it as a shared address period.


Step 3: Calculate the total number of shared address periods for the applicant.

Step 4: Check if the total number of shared address periods for the applicant is at least 7 out of the last 10 years. If yes, the applicant is eligible for a resident permit.

Step 5: Output the decision, i.e., whether the applicant is eligible or not.

Here is the pseudo code for the decision model:

css

 Copy code

```
shared_periods = []
for period in applicant_periods:
    if period in marriage_periods:
        spouse_period = find_period_in_spouse_periods(period)
        if spouse_period is not None:
            shared_period = calculate_shared_period(period, spouse_period)
            shared_periods.append(shared_period)

total_shared_periods = sum(shared_periods)
if total_shared_periods >= 7:
    output("Eligible for resident permit")
else:
    output("Not eligible for resident permit")
```

In the above pseudo code, `applicant_periods` refers to the list of periods the applicant lived at different addresses, `marriage_periods` refers to the list of periods of the applicant's marriage, `spouse_periods` refers to the list of periods the spouse lived at different addresses, `find_period_in_spouse_periods()` function takes a period and returns the corresponding period in the `spouse_periods` list, `calculate_shared_period()` function takes two periods and returns the period of shared residence between them, `shared_periods` refers to the list of periods of shared residence between the applicant and the spouse, `total_shared_periods` is the total number of shared residence periods between the applicant and the spouse.

Request:

Did you forget the last 10 years' requirement?

Response:

You are right. I apologize for the oversight. To ensure that the eligibility criteria of living at the same address for at least 7 of the last 10 years is met, we need to modify the decision model. We can modify the pseudo code as follows:

```
current_date = today's date
eligible_periods = []
for period in applicant_periods:
    if period in marriage_periods:
        spouse_period = find_period_in_spouse_periods(period)
        if spouse_period is not None:
            shared_period = calculate_shared_period(period, spouse_period)
            if shared_period[1] >= current_date - timedelta(days=3650):
                eligible_periods.append(shared_period)

total_eligible_years = sum([calculate_years(period) for period in eligible_periods])
if total_eligible_years >= 7:
    output("Eligible for resident permit")
else:
    output("Not eligible for resident permit")
```

In the above pseudo code, we have introduced a new variable `current_date`, which is the current date. We then check if the end date of the shared period is within the last 10 years from the `current_date`. If yes, we add that shared period to the `eligible_periods` list. We then calculate the total number of years in the `eligible_periods` list and check if it is greater than or equal to 7. If yes, the applicant is eligible for a resident permit.

The `timedelta(days=3650)` represents a time delta of 10 years in days. The `calculate_years()` function is used to calculate the number of years in a period, and `eligible_periods` refers to the list of eligible shared periods between the applicant and the spouse.

Request:

Can you show this solution in Java?

Response:



Sure, here is an example implementation of the decision model in Java:

```
java Copy code
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class ResidentPermitEligibility {

    public static void main(String[] args) {
        // Input data
        List<Period> applicantPeriods = new ArrayList<>();
        applicantPeriods.add(new Period(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 1, 1)));
        applicantPeriods.add(new Period(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 1, 1)));
        applicantPeriods.add(new Period(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 1, 1)));

        List<Period> spousePeriods = new ArrayList<>();
        spousePeriods.add(new Period(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 1, 1)));
        spousePeriods.add(new Period(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 1, 1)));
        spousePeriods.add(new Period(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 1, 1)));

        List<Period> marriagePeriods = new ArrayList<>();
        marriagePeriods.add(new Period(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 1, 1)));
        marriagePeriods.add(new Period(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 1, 1)));

        // Decision model
        LocalDate currentDate = LocalDate.now();
```

Request:

The code is not complete. Please show the complete solution in Java.

Response:

I apologize for that. Here's the complete solution in Java:

I copied the complete produced code:

```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.List;

public class ResidentPermitEligibility {

    public static void main(String[] args) {
        // Input data
        List<Period> applicantPeriods = new ArrayList<>();
        applicantPeriods.add(new Period(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 12, 31), "123 Main St, Anytown, USA"));
        applicantPeriods.add(new Period(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 12, 31), "456 Oak St, Anytown, USA"));
        applicantPeriods.add(new Period(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 3, 4), "789 Elm St, Anytown, USA"));

        List<Period> spousePeriods = new ArrayList<>();
        spousePeriods.add(new Period(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 12, 31), "123 Main St, Anytown, USA"));
        spousePeriods.add(new Period(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 12, 31), "120 Maple St, Anytown, USA"));
        spousePeriods.add(new Period(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 3, 4), "789 Elm St, Anytown, USA"));

        List<Period> marriagePeriods = new ArrayList<>();
        marriagePeriods.add(new Period(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 12, 31)));
        marriagePeriods.add(new Period(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 3, 4)));

        // Decision model
        LocalDate currentDate = LocalDate.now();
```

```

List<Period> eligiblePeriods = new ArrayList<>();
for (Period period : applicantPeriods) {
    if (marriagePeriods.contains(period)) {
        Period spousePeriod = findPeriodInSpousePeriods(period, spousePeriods);
        if (spousePeriod != null) {
            Period sharedPeriod = calculateSharedPeriod(period, spousePeriod);
            if (sharedPeriod.getTo().isAfter(currentDate.minusYears(10))) {
                eligiblePeriods.add(sharedPeriod);
            }
        }
    }
}
int totalEligibleYears = 0;
for (Period period : eligiblePeriods) {
    totalEligibleYears += calculateYears(period);
}
if (totalEligibleYears >= 7) {
    System.out.println("Eligible for resident permit");
} else {
    System.out.println("Not eligible for resident permit");
}
}

public static Period findPeriodInSpousePeriods(Period period, List<Period> spousePeriods) {
    for (Period spousePeriod : spousePeriods) {
        if (spousePeriod.equals(period)) {
            return spousePeriod;
        }
    }
    return null;
}

public static Period calculateSharedPeriod(Period period1, Period period2) {
    LocalDate from = period1.getFrom().isAfter(period2.getFrom()) ? period1.getFrom() : period2.getFrom();
    LocalDate to = period1.getTo().isBefore(period2.getTo()) ? period1.getTo() : period2.getTo();
}

```

....

The code was abrupt.

Could you try to execute your code and show the results?

As an AI language model, I don't have a local environment to execute the code. However, I can show you an example of how to use the `isEligibleForResidentPermit` method with the provided input data:

```
less Copy code
public static void main(String[] args) {
    // Creating the list of periods living at an address for the applicant
    List<AddressPeriod> applicantPeriods = new ArrayList<>();
    applicantPeriods.add(new AddressPeriod(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 1, 1)));
    applicantPeriods.add(new AddressPeriod(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 1, 1)));
    applicantPeriods.add(new AddressPeriod(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 1, 1)));

    // Creating the list of periods living at an address for the spouse
    List<AddressPeriod> spousePeriods = new ArrayList<>();
    spousePeriods.add(new AddressPeriod(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 1, 1)));
    spousePeriods.add(new AddressPeriod(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 1, 1)));
    spousePeriods.add(new AddressPeriod(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 1, 1)));

    // Creating the list of marriage periods
    List<MarriagePeriod> marriagePeriods = new ArrayList<>();
    marriagePeriods.add(new MarriagePeriod(LocalDate.of(2010, 1, 1), LocalDate.of(2020, 1, 1)));
    marriagePeriods.add(new MarriagePeriod(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 1, 1)));

    // Checking if the applicant is eligible for a resident permit
    boolean isEligible = isEligibleForResidentPermit(applicantPeriods, spousePeriods, marriagePeriods);
    System.out.println("Is the applicant eligible for a resident permit? " + isEligible);
}
```

Assuming the implementation of the `isEligibleForResidentPermit` method is correct, this code should output: "Is the applicant eligible for a resident permit? true".

This is an invalid answer, as even a manual evaluation of the Challenge's lists shows that this applicant is NOT eligible for a resident permit. Really the eligible periods are:

[2013-03-04; 2015-12-31] 1032 days

[2021-01-01; 2023-03-04] 792 days

Total: 1824 days ~ 4.9 years

Then I asked:

Could you show this decision model in DMN?

Certainly! Here's a DMN decision model that represents the logic for determining if an applicant is eligible for a resident permit based on the provided input data:

The image you are requesting does not exist or is no longer available.

imgur.com

ChatGPT found a link to an old image (probably a previously generated DMN diagram) but that link was broken.

It didn't make a lot of sense to continue this dialog. Still, instead of giving up, I decided to clean up the produced Java code and run it.

PART 2. Validation of the ChatGPT Solution

I created a simple Java project PermitEligibilityChatGPT in Eclipse and copied the above code into the Java class PermitEligibility. After cleaning up some obvious syntax errors, I came up with this “good-looking” Java code shown below. So, assuming that the following code was completely generated by ChatGPT, we will analyze it.

```
package permit.eligibility;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class PermitEligibility {
    .....
    private static final int REQUIRED_YEARS = 7;
    private static final int LOOKBACK_YEARS = 10;
    .....

    public static boolean isEligible(List<Residence> applicantResidences, List<Residence> spouseResidences,
    ..... List<Marriage> marriages) {
    ..... LocalDate today = LocalDate.now();
    ..... LocalDate lookbackDate = today.minusYears(LOOKBACK_YEARS);
    .....

    ..... // Find all addresses where applicant and spouse lived together during marriage periods
    ..... List<String> sharedAddresses = new ArrayList<>();
    ..... for (Marriage marriage : marriages) {
    .....     LocalDate marriageStart = marriage.getStart();
    .....     LocalDate marriageEnd = marriage.getEnd() != null ? marriage.getEnd() : today;
    .....     for (Residence applicantResidence : applicantResidences) {
    .....         if (applicantResidence.contains(marriageStart) && applicantResidence.contains(marriageEnd)) {
    .....             for (Residence spouseResidence : spouseResidences) {
    .....                 if (spouseResidence.contains(marriageStart) && spouseResidence.contains(marriageEnd)) {
    .....                     if (applicantResidence.getAddress().equals(spouseResidence.getAddress())) {
    .....                         sharedAddresses.add(applicantResidence.getAddress());
    .....                         break;
    .....                     }
    .....                 }
    .....             }
    .....         }
    .....     }
    ..... }
    ..... }
    ..... }
    ..... }
```

Here we can see that “sharedAddresses” are those for which marriage periods cover residence periods for both the applicant and the spouse. The periods that are only partially covered will be ignored.

```
.....// Check if shared addresses were lived in for at least 7 of the last 10 years
.....int eligibleYears = 0;
.....for (String address : sharedAddresses) {
.....    int livedYears = 0;
.....    for (Residence applicantResidence : applicantResidences) {
.....        if (applicantResidence.getAddress().equals(address) && applicantResidence.getStart().isAfter(lookbackDate)) {
.....            livedYears += applicantResidence.getDurationYears();
.....        }
.....    }
.....    for (Residence spouseResidence : spouseResidences) {
.....        if (spouseResidence.getAddress().equals(address) && spouseResidence.getStart().isAfter(lookbackDate)) {
.....            livedYears += spouseResidence.getDurationYears();
.....        }
.....    }
.....    System.out.println("livedYears = " + livedYears);
.....    if (livedYears >= REQUIRED_YEARS) {
.....        eligibleYears += livedYears;
.....    }
.....}
.....
.....System.out.println("eligibleYears = " + eligibleYears);
.....return eligibleYears >= REQUIRED_YEARS;
.....}
.....
```

Note that “eligibleYears” can be calculated twice, the first time for the applicant and then for the spouse. That’s certainly wrong.

Here are supporting inner classes generated by ChatGPT:

```

----private static class Residence {
-----private LocalDate start;
-----private LocalDate end;
-----private String address;
-----
-----public Residence(LocalDate start, LocalDate end, String address) {
-----    this.start = start;
-----    this.end = end;
-----    this.address = address;
-----}
-----
-----public boolean contains(LocalDate date) {
-----    return !date.isBefore(start) && (end == null || !date.isAfter(end));
-----}
-----
-----public String getAddress() {
-----    return address;
-----}
-----
-----public int getDurationYears() {
-----    if (end == null) {
-----        return LocalDate.now().getYear() - start.getYear();
-----    } else {
-----        return end.getYear() - start.getYear();
-----    }
-----}
-----
-----public LocalDate getStart() {
-----    return start;
-----}
----}

----private static class Marriage {
-----private LocalDate start;
-----private LocalDate end;
-----
-----public Marriage(LocalDate start, LocalDate end) {
-----    this.start = start;
-----    this.end = end;
-----}
-----
-----public LocalDate getStart() {
-----    return start;
-----}
-----
-----public LocalDate getEnd() {
-----    return end;
-----}
----}

```

And finally here is a test case correctly generated by ChatGPT based on the Challenge's lists:

```
public static void main(String[] args) {
    // Creating the list of periods living at an address for the applicant
    List<Residence> applicantPeriods = new ArrayList<>();
    applicantPeriods.add(new Residence(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 12, 31), "123 Main St, Anytown, USA"));
    applicantPeriods.add(new Residence(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 12, 31), "456 Oak St, Anytown, USA"));
    applicantPeriods.add(new Residence(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 3, 4), "789 Elm St, Anytown, USA"));

    // Creating the list of periods living at an address for the spouse
    List<Residence> spousePeriods = new ArrayList<>();
    spousePeriods.add(new Residence(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 12, 31), "123 Main St, Anytown, USA"));
    spousePeriods.add(new Residence(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 12, 31), "120 Maple St, Anytown, USA"));
    spousePeriods.add(new Residence(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 3, 4), "789 Elm St, Anytown, USA"));

    // Creating the list of marriage periods
    List<Marriage> marriagePeriods = new ArrayList<>();
    marriagePeriods.add(new Marriage(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 12, 31)));
    marriagePeriods.add(new Marriage(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 3, 4)));

    // Checking if the applicant is eligible for a resident permit
    boolean isEligible = isEligible(applicantPeriods, spousePeriods, marriagePeriods);
    System.out.println("Is the applicant eligible for a resident permit? " + isEligible);
}
```

Then I added printing for “livedYears” and “eligibleYears” and ran this code “as is”. Here are the produced results:

```
livedYears = 0
livedYears = 4
eligibleYears = 0
Is the applicant eligible for a resident permit? false
```

This result is correct as we did expect ineligibility as well. However, it was produced based on false logic.

Obviously, something is wrong with the above “good-looking” code. A quick analysis of the above code shows several serious issues:

- dealing with years rather than with days inside different periods
- Residence periods do not always have to be inside Marriage periods, they may intersect
- double calculation of estimated years
- ignoring less than 1-year periods.

A partial correction of this code would not be productive, so I decided to essentially redesign it.

PART 3. Manual Conversion of ChatGPT Solution to Working Java Code

Here is the essentially modified code. I created 3 java classes: Period.java, Residence.java, and Problem.java.

Period.java

```
package permit.eligibility;

import java.time.Duration;
import java.time.LocalDate;

public class Period {
    private LocalDate start;
    private LocalDate end;

    public Period(LocalDate start, LocalDate end) {
        this.start = start;
        this.end = end;
    }

    public int getDurationDays() {
        Duration duration = Duration.between(start.atStartOfDay(), end.atStartOfDay());
        return (int) duration.toDays();
    }

    public Period getIntersection(Period p) {
        LocalDate intersectionStart = this.start.isAfter(p.start) ? this.start : p.start;
        LocalDate intersectionEnd = this.end.isBefore(p.end) ? this.end : p.end;

        if (intersectionStart.isAfter(intersectionEnd)) {
            return null; // no intersection
        } else {
            return new Period(intersectionStart, intersectionEnd);
        }
    }
}
```

Residence.java

```
package permit.eligibility;

import java.time.LocalDate;

public class Residence {
    ....Period period;
    ....String address;
    ....
}

public Residence(LocalDate start, LocalDate end, String address) {
    ....this.period = new Period(start,end);
    ....this.address = address;
    ....
}

public Period getPeriod() {
    ....return period;
    ....
}

public String getAddress() {
    ....return address;
    ....
}
}
```

Problem.java

```
package permit.eligibility;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class Problem {

    ....static final int REQUIRED_YEARS = 7;
    ....static final int LOOKBACK_YEARS = 10;
    ....LocalDate applicationDate = LocalDate.now();
    ....LocalDate lookbackDate = applicationDate.minusYears(LOOKBACK_YEARS);
    ....Period consideredPeriod = new Period(lookbackDate, applicationDate);
    ....
}

public Problem(LocalDate applicationDate, int lookbackYears) {
    ....LocalDate lookbackDate = applicationDate.minusYears(lookbackYears);
    ....consideredPeriod = new Period(lookbackDate, applicationDate);
    ....
}
```

```

public boolean isEligible(List<Residence> applicantResidences,
.....
..... List<Residence> spouseResidences,
.....
..... List<Period> marriagePeriods) {

.....// Find all intersection periods when applicant and spouse lived at the same address
.....List<Period> sharedPeriods = new ArrayList<>();
.....for (Residence applicantResidence : applicantResidences) {
.....for (Residence spouseResidence : spouseResidences) {
.....if (applicantResidence.getAddress().equals(spouseResidence.getAddress())) {
.....Period intersection = applicantResidence.getPeriod().getIntersection(spouseResidence.getPeriod());
.....if (intersection != null) {
.....sharedPeriods.add(intersection);
.....}
.....}
.....}
.....}

.....int eligibleDays = 0;
.....for (Period marriagePeriod : marriagePeriods) {
.....for (Period sharedPeriod : sharedPeriods) {
.....Period intersection = marriagePeriod.getIntersection(sharedPeriod);
.....if (intersection == null)
.....continue;
.....intersection = intersection.getIntersection(consideredPeriod);
.....if (intersection != null) {
.....System.out.println("sharedPriod: [" + intersection.start + ";" + intersection.end + "]" + " - "
.....+ intersection.getDurationDays() + " days");
.....eligibleDays += intersection.getDurationDays();
.....}
.....}
.....}
.....System.out.println("eligibleDays = " + eligibleDays);
.....return eligibleDays >= REQUIRED_YEARS * 365;
.....}

```



```

... public static void main(String[] args) {
.....
..... LocalDate applicationDate = LocalDate.of(2023, 3, 4);
..... Problem problem = new Problem(applicationDate, 10);
.....
..... // TEST 1
..... // Creating the list of periods living at an address for the applicant
..... List<Residence> applicantPeriods = new ArrayList<>();
..... applicantPeriods.add(new Residence(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 12, 31), "123 Main St, Anytown, USA"));
..... applicantPeriods.add(new Residence(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 12, 31), "456 Oak St, Anytown, USA"));
..... applicantPeriods.add(new Residence(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 3, 4), "789 Elm St, Anytown, USA"));

..... // Creating the list of periods living at an address for the spouse
..... List<Residence> spousePeriods = new ArrayList<>();
..... spousePeriods.add(new Residence(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 12, 31), "123 Main St, Anytown, USA"));
..... spousePeriods.add(new Residence(LocalDate.of(2016, 1, 1), LocalDate.of(2020, 12, 31), "120 Maple St, Anytown, USA"));
..... spousePeriods.add(new Residence(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 3, 4), "789 Elm St, Anytown, USA"));

..... // Creating the list of marriage periods
..... List<Period> marriagePeriods = new ArrayList<>();
..... marriagePeriods.add(new Period(LocalDate.of(2010, 1, 1), LocalDate.of(2015, 12, 31)));
..... marriagePeriods.add(new Period(LocalDate.of(2021, 1, 1), LocalDate.of(2023, 3, 4)));

..... // Checking if the applicant is eligible for a resident permit
..... boolean isEligible = problem.isEligible(applicantPeriods, spousePeriods, marriagePeriods);
..... System.out.println("TEST 1: Is the applicant eligible for a resident permit?" + isEligible);

```

This main test is basically the same as in ChatGPT' code just I parameterized "applicationDate" and "lookbackYears".

My execution results:

```

sharedPriod: [2013-03-04;2015-12-31] 1032 days
sharedPriod: [2021-01-01;2023-03-04] 792 days
eligibleDays = 1824
TEST 1: Is the applicant eligible for a resident permit? false

```