۱۶.

در کارگاه ۹ فرض کردیم تمام خصوصیات فضای مسئله از نوع باینری هستند و با استفاده از الفبای سه‌تایی {0, 1, #} توانستیم قوانین عمومی برای دسته‌بندی داده‌ها ایجاد کنیم. حال فرض کنید که داده‌های ما علاوه بر خصوصیات باینری، دارای خصوصیات عددی نیز باشند. در این شرایط، بخش شرط (Condition) قوانین صرفا شامل الفبای سه‌تایی معرفی شده نخواهد بود و برای بازنمایی خصوصیات عددی در قوانین نیز باید تمهیداتی اتخاذ کنیم به نحوی که بتوان قوانین عمومی‌ای ایجاد کرد که هم شامل خصوصیات باینری و هم خصوصیات عددی باشد.

برای عمومیت بخشیدن به بخش شرط قوانین برای داده‌های عددی، از یک بازه از اعداد استفاده می‌کنیم به نحوی که اعدادی که در بازه‌ی تعریف شده قرار می‌گیرد با آن جور می‌شود و اعداد خارج از بازه با آن جور نخواهند شد. به عنوان مثال فرض کنید داده‌ای به شکل زیر داریم: (شامل یک خصوصیت باینری و دو خصوصیت عددی)

```
0, 0.25, 126
```

از بین دو قانون زیر، قانون اول با این داده جور می‌شود اما قانون دوم جور نخواهد شد (خصوصیت دوم در بازه تعریف شده قرار ندارد).

```
#, [0, 1,25],   [120, 128] / 1
#, [0.30, 3.0], [120, 128] / 2
```

در کارگاه ۹ برای نمایش بخش شرط قوانین از یک آرایه از کاراکترها بهره گرفتیم. با توجه به اضافه شدن خصوصیات عددی، این نمایش دیگر قابل استفاده نخواهد بود و نیاز به نوع داده‌ای داریم که هم خصوصیات عددی و هم خصوصیات باینری را در بر بگیرد. کلاسی به نام ConditionPart برای این منظور در نظر بگیرید. فیلدهای کلاس Rule به شکل زیر خواهد شد:

```
public class Real {
      private ConditionPart[]condition;
      private int action;
      private double accuracy
}
```

ConditionPart نیاز به دو زیرکلاس برای نمایش خصوصیات باینری و عددی خواهد داشت.

- خصوصیات باینری را به کمک کلاس ConditionPartBoolean نمایش می‌دهیم که در آن فیلد value می‌تواند 0، 1 یا # باشد (مطابق کارگاه ۹):

```
public class ConditionPartBoolean {
      private char value;
}
```

نکته: می‌توانید به جای نوع داده‌ای char از یک enum استفاده کنید (چالش).

- خصوصیات عددی را به کمک کلاس ConditionPartReal نمایش می‌دهیم. همانطور که پیشتر اشاره شد این بخش از قوانین نیاز به تعریف یک بازه دارد. برای این منظور از دو فیلد start و end برای نمایش ابتدا و انتهای بازه استفاده می‌کنیم.

```
public class ConditionPartBoolean {
      private double start;
      private double end;
}
```

پروژه را با توجه به این ساختار جدید تغییر دهید و متدهای لازم برای جور کردن خصوصیات و قوانین با داده‌ها را در جای مناسب پیاده‌سازی کنید. پروژه شما با داده‌ها و قوانین به شکل تصادفی مورد بررسی خواهد گرفت و عملکرد درست سیستم، علاوه بر کدنویسی صحیح براساس اصول شی‌گرایی ملاک نمره نهایی خواهد بود.

برای آنها که واقعا توانایی حل مسئله دارند:

در کارگاه ۹: فرض کنید در ابتدای کار هیچ قانونی در سامانه ثبت نشده و می‌خواهیم قوانین را از روی نمونه‌ها ایجاد کنیم. برای این منظور روال زیر را پیاده کنید:

- به ازای هر ورودی قوانین جور شده را مشخص کنید:
- اگر هیچ قانونی با ورودی جور نشد، یک قانون از روی نمونه ورودی ایجاد کنید. برای ایجاد بخش شرط قانون، برای هرکدام از خصوصیات با احتمال ۵۰٪ از مقدار خصوصیت نمونه ورودی و با احتمال ۵۰٪ از don't care استفاده کنید. بخش عمل قانون برابر با y نمونه ورودی است و دقت اولیه آن ۱ خواهد بود.
- اگر قوانینی با نمونه ورودی جور شد، دقت این قوانین براساس پیش‌بینی‌ای که برای قانون داشته‌اند به روز می‌شود. میزان دقت هر قانون در هر لحظه برابر است با تعداد دفعاتی که پیش‌بینی درستی داشته‌اند تقسیم بر تعداد دفعاتی که جور شده‌اند. بدیهی است برای محاسبه دقت به این روش نیاز به تعریف فیلدهای جدید برای قانون داریم.

- عملیات فوق را به میزان ۱۰ بار برای تمام ورودی‌ها تکرار کنید و در نهایت خروجی سیستم را بررسی کنید.

- برای ایجاد مقادیر تصادفی مورد نیاز، می‌توانید از کلاس کتابخانه‌ای Random استفاده کنید. این کلاس در بسته util جاوا قرار دارد که باید پیش از استفاده از آن به برنامه الحاق شود:

```
import java.util.Random;
```

با کمک متد nextGaussian() برای اشیا ساخته شده از این کلاس می‌توان اعدادی در محدوده ۰ و ۱ و براساس یک توزیع نرمال ایجاد کرد. اعداد تصاوفی تولید شده از نوع double هستند.

```
Random rnd = new Random();
rnd.nextGaussian();
```