conascenza

nascere e crescere insieme

cos’è

si dice conascenza quando due o più elementi sono legati tra di loro, ovvero quando dobbiamo cambiare gli altri elementi alla modifica di uno solo di questi

classificazione

- grado: numero di entità collegate tra di loro, più è alto più sarà difficile rimuovere la conascenza

- posizione: distanza a livello di astrazione tra le entità collegate, per esempio collegamento tra modello e modello è più facile da rimuovere rispetto ad un collegamento tra presentation e l’implementazione di un proxy

- forza: quanto è forte il collegamento e di conseguenza quanto sforzo serve per rimuovere questo accoppiamento

ulteriore divisione

- statici: si possono trovare semplicemente vedendo il codice senza bisogno di eseguirlo

- dinamici: si riescono a scoprire solamente eseguendo il programma

nome ✅

```
const prescriptions = ref<Prescription[]>([]);

const fetchPrescriptions = async () => {
  const { data } = await apolloQueryToGetPrescriptions();
  prescriptions.value = data.prescriptions.nodes;
};

const populateTable = (): DataRow[] => {
  return prescriptions.value.map(prescription => {/* ... */});
};

const prescriptionExistsByCode = (code: string): boolean => {
  return prescriptions.value.some(p => p.prescriptionCode === code);
};
```

tipo ✅

```
const props = defineProps<{
  currentDate?: DateTime;
}>();

const beginDate = ref<DateTime>(props.currentDate ?? DateTime.now());
```

significato

prima

```
const weightUnitOfMeasure = ref<weightUnitsOfMeasure>('kg');

// <InputSelect
//   v-model="weightUnitOfMeasure"
//   :options="[
//     { value: 'kg', label: 'Chilogrammi' },
//     { value: 'g', label: 'Grammi' },
//   ]"
//   class="flex-1"
// />

const calculateBmi = (weightType: string, heightType: string, weig
  if (!weight || !height) return 0;

  const denW = weightType === 'kg' ? 1 : 1000;
  const numH = heightType === 'cm' ? 1 : 100;

  return (weight / denW) / Math.pow((height * numH) / 100, 2);
};
```

dopo

```
enum WeightUnit { Kg, G }
const weightUnitOfMeasure = ref<WeightUnit>(WeightUnit.Kg);

// <InputSelect
//   v-model="weightUnitOfMeasure"
//   :options="[
//     { value: WeightUnit.Kg, label: 'Chilogrammi' },
//     { value: WeightUnit.G, label: 'Grammi' },
//   ]"
//   class="flex-1"
// />

const calculateBmi = (weightType: WeightUnit, heightType: string,
  if (!weight || !height) return 0;

  const denW = weightType === WeightUnit.Kg ? 1 : 1000;
  const numH = heightType === 'cm' ? 1 : 100;

  return (weight / denW) / Math.pow((height * numH) / 100, 2);
};
```

algoritmo

prima

```
const cols: TableColumn[] = [
  { title: 'Paziente' },
  { title: 'Codice Fiscale' },
  ...(props.excludeMonitoringColumn ? [] : [{ title: 'Monitoraggio' }]),
  { title: 'Azioni' },
];
```

```
const dataRow: DataRow = {
  cells: [
    patientAvatarDataCell,
    { mainText: p.child.taxCode },
    ...(props.excludeMonitoringColumn ? [] : [diaryMonitoringDataCell]),
    {
      btns: [/* ... */],
    },
  ],
};
```

dopo

```
const cols: TableColumn[] = [
  { title: 'Paziente', id: Column.patient },
  { title: 'Codice Fiscale', id: Column.taxCode },
  ...(props.excludeMonitoringColumn ? [] : [{ title: 'Monitoraggio', id: Column.monitoring }]),
  { title: 'Azioni', id: Column.actions},
];
```

```
const dataRow: DataRow = { cells: [] };
for (const column of cols) {
  switch (column.id) {
    case Column.patient:
      dataRow.cells.push(/* ... */);
      break;
    case Column.taxCode:
      dataRow.cells.push(/* ... */);
      break;
    case Column.monitoring:
      dataRow.cells.push(/* ... */);
      break;
    case Column.actions:
      dataRow.cells.push(/* ... */);
      break;
  }
}
```

posizione

prima

```
const calculateOccurenciesInTimeframe = (
  startingDateSeconds: number,
  periodicity: number,
  paiDate: number,
  maxOutpatientDate: number,
  outpatient?: OutpatientServiceParamInput
) => {/* ... */};
```

dopo

```
const calculateOccurenciesInTimeframe = ({
  eventStartDate,
  eventPeriodicity,
  timeframeStartDate,
  timeframeEndDate,
  outpatient,
}: {
  eventStartDate: number;
  eventPeriodicity?: number;
  timeframeStartDate: number;
  timeframeEndDate: number;
  outpatient?: OutpatientServiceParamInput;
}) => {/* ... */};
```

valore

prima

```
class Region {
  private istatCode: number;

  constructor(istatCode: number) {
    this.istatCode = istatCode;
  }
}

const veneto = new Region(213242);
```

dopo

```
enum IstatRegionCode {
  ABRUZZO = 13,
  // ...
  VENETO = 5,
}

class Region {
  private istatCode: IstatRegionCode;

  constructor(istatCode: IstatRegionCode) {
    this.istatCode = istatCode;
  }
}

const veneto = new Region(IstatRegionCode.VENETO);
```

qualche domanda?

grazie

d.dalbosco@elty.it