Exercise 1: Student Attendance & Eligibility System

Design

- Daily attendance is stored as int because attendance count is always a whole number.
- Total classes and attended classes are int.
- Attendance percentage is calculated using double to handle fractions.
- Eligibility is displayed as an int percentage.

Code:

```
int totalClasses = 90;
int attendedClasses = 72;
double percentage = (attendedClasses * 100.0) / totalClasses;
int displayPercentage = (int)Math.Round(percentage);
```

Explanation

- 100.0 forces floating-point division.
- Math.Round is used instead of direct casting to avoid unfair loss of value.
- Truncation (int)percentage would cut off decimals (79.9 → 79), which may wrongly mark a student ineligible.
- Rounding gives a fair academic result.

Exercise 2: Online Examination Result Processing

Design

- Marks per subject are int.
- Average is stored as double for decimal precision.
- Scholarship check requires converting average to int.

Code:

```
int m1 = 78, m2 = 85, m3 = 90;
double average = (m1 + m2 + m3) / 3.0;
double formattedAverage = Math.Round(average, 2);
int scholarshipScore = (int)Math.Floor(formattedAverage);
```

Explanation

- double keeps two decimal places for result display.
- Converting to int causes precision loss.
- Floor is safer than rounding up to prevent giving undeserved eligibility.
- Example: 79.99 becomes 79, not 80.

Exercise 3: Library Fine Calculation System

Design

- Fine per day is decimal to avoid money precision errors.
- Days overdue is int.
- Total fine is decimal.
- Analytics logging uses double.

Code:

```
decimal finePerDay = 2.50m;
int daysLate = 4;
decimal totalFine = finePerDay * daysLate;
double logValue = (double)totalFine;
```

Explanation

- decimal is best for currency calculations.
- double is acceptable for analytics where slight precision loss is okay.
- Explicit casting is required because decimal to double is not implicit.

Exercise 4: Banking Interest Calculation Module

Design

- Balance is decimal.
- Interest rate from API is float.
- Interest calculation is done using decimal.

Code:

```
decimal balance = 10000m;
float rate = 7.5f;
decimal interest = balance * ((decimal)rate / 100);
balance += interest;
```

Explanation

- float cannot be directly multiplied with decimal.
- Explicit conversion avoids mixing floating-point inaccuracies with money.
- Implicit conversion fails because decimal is higher precision than float.

Exercise 5: E-Commerce Order Pricing Engine

Design

- Cart total is double due to multiple calculations.
- Tax and discount use decimal.
- Final payable amount is decimal.

Code:

```
double cartTotal = 1999.99;
decimal taxRate = 0.18m;
decimal finalAmount = (decimal)cartTotal * (1 + taxRate);
```

Explanation

- double is fast but risky for money.
- Conversion happens once at the final stage.
- Storing final value in decimal ensures billing accuracy.
- Frequent double-decimal conversions are avoided.

Exercise 6: Weather Monitoring & Reporting

Design

- Sensor readings are short.
- Converted temperature is stored as double.
- Dashboard shows int value.

Code:

```
short rawValue = 320;
double celsius = rawValue / 10.0;
int displayTemp = (int)Math.Round(celsius);
```

Explanation

- short saves memory for sensor data.
- Conversion to double avoids overflow during calculations.
- Explicit cast to int with rounding improves readability.
- Direct casting without rounding may misrepresent temperature.

Exercise 7: University Grading Engine

Design

- Final score is double.
- Grades are stored as byte.

Code:

```
double score = 82.5;
byte grade;
if (score >= 90) grade = 10;
else if (score >= 80) grade = 9;
else if (score >= 70) grade = 8;
else grade = 5;
```

Explanation

- Direct casting from double to byte is unsafe.
- Validation ensures grade remains within 0–255.
- Code:al mapping avoids overflow and invalid grade values.

Exercise 8: Mobile Data Usage Tracker

Design

- Usage is stored in bytes as long.
- Displayed in MB and GB using double.
- Monthly summary is rounded to int.

Code:

```
long bytesUsed = 5368709120;
double gbUsed = bytesUsed / (1024.0 * 1024 * 1024);
int summary = (int)Math.Round(gbUsed);
```

Explanation

- long prevents overflow for large byte values.
- Implicit conversion to double is safe.
- Rounding gives user-friendly numbers.
- Truncation would underreport usage.

Exercise 9: Warehouse Inventory Capacity Control

Design

- Item count is int.
- Max capacity is ushort.

Code:

```
int currentStock = 500;
ushort maxCapacity = 600;
bool isFull = currentStock >= maxCapacity;
```

Explanation

- ushort avoids negative capacity.
- Implicit conversion to int during comparison is safe.
- Reverse conversion could cause overflow if not checked.
- Signed vs unsigned mismatch is handled carefully.

Exercise 10: Payroll Salary Computation

Design

- Basic salary is int.
- Allowances and deductions are double.
- Net salary is decimal.

Code:

```
int basic = 30000;
double allowance = 4500.75;
double deduction = 1200.25;
decimal netSalary = basic + (decimal)allowance - (decimal)deduction;
```

Explanation

- int is enough for fixed salary.
- double handles percentage-based calculations.
- Final salary stored in decimal ensures financial accuracy.
- Explicit casting avoids silent precision errors.