# Custom type conversions and implicit conversions

```
/*

        Topics:
            . Type conversion operators
            . Implicit conversions

*/
```

# Custom type conversions

```
export class Number
{

public:
  Number() = default;
  Number(int value);

  //Number to double
  explicit operator double() const { return (static_cast<double>(m_wrapped_int)); }

  //Number to Point
  explicit operator Point() const
  {
    return Point(static_cast<double>(m_wrapped_int), static_cast<double>(m_wrapped_int));
  }

  // getter
  int get_wrapped_int() const { return m_wrapped_int; }

  ~Number();

private:
  int m_wrapped_int{ 0 };
};
```

# Custom type conversions

```
export double sum(double a, double b) {
  return a + b;
}
void use_point(const Point &p) {
  std::cout << "Printing the point from use_point func: " << p << "\n";
}

int main(){

  Number n1(22);
  Number n2(10);

  double result = sum(static_cast<double>(n1),static_cast<double>(n2));
  std::cout << "result: " << result << "\n";
  //use_point(static_cast<Point>(n1)); // This uses the custom conversion
                                       // operator
}
```

# Implicit conversions

```
//If a binary operator is implemented as a member, the first operand can't participate
// in implicit conversions
  Number n1(22);
  auto n12 = (n1 + 2); // 2 is implicitly converted
  auto n21 = (2 + n1)  // 2 can't be implicitly converted
```

# Implicit conversions

```
export class Number
{
  friend std::ostream &operator<<(std::ostream &out, const Number &number);
  // Arithmetic operators
  //friend Number operator+(const Number &left_operand, const Number &right_operand);
  friend Number operator-(const Number &left_operand, const Number &right_operand);
  friend Number operator*(const Number &left_operand, const Number &right_operand);
  friend Number operator/(const Number &left_operand, const Number &right_operand);
  friend Number operator%(const Number &left_operand, const Number &right_operand);

public:
  Number(int value);

  //Implicit conversions won't work for the first operand
  Number operator+(const Number& right) const{
      return Number(m_wrapped_int + right.m_wrapped_int);
  }

  // getter
  int get_wrapped_int() const { return m_wrapped_int; }

private:
  int m_wrapped_int{ 0 };
};
```