

# インスタンス生成時の問題点

- クラスからインスタンスを生成した直後だと、  
メンバ変数は初期化されていない
  - メンバ変数を使ったプログラムを記述したとき、  
初期化忘れて誤動作の可能性がある
- (例) Sample301のCarクラスのspeedメンバに  
値を格納しなくてもdrive関数が実行可能
  - 想定外の数値が出力(バグの原因)

# コンストラクタとデストラクタ

- **コンストラクタ(constructor)**

どのようなクラスであってもインスタンス生成時に  
必ず実行される関数

コンストラクタの中でメンバ変数の初期化処理を  
入れておけばよい

関数名はクラス名と同じにする

例: **Car**クラスのコンストラクタは **Car()**

# コンストラクタとデストラクタ

- **デストラクタ(destructor)**  
インスタンスが破棄される直前に必ず実行される関数

関数名はクラス名の前に「~(チルダ)」を付ける

例: **Car**クラスのデストラクタ: **~Car()**

# コンストラクタとデストラクタ

- 教科書P134~136 Sample401
- Sample301cフォルダからSample401フォルダを作成  
`robocopy Sample301c Sample401`  
`cd Sample401`
- `car.h`, `car.cpp`, `main.cpp`を編集

# コンストラクタとデストラクタ

- car.h (Sample401)

```
#pragma once
class Car {
public:
    Car();
    ~Car();
    void setSpeed(double speed);
    double getSpeed();
    double getMigration();
    void drive(double hour);
private:
    double m_speed;
    double m_migration;
};
```

# コンストラクタとデストラクタ

- car.cpp (Sample401)

```
#include "car.h"
#include <iostream>
using namespace std;

Car::Car() : m_speed(0), m_migration(0) {
    cout << "Carクラスのインスタンス生成" << endl;
}
Car::~~Car() {
    cout << "Carクラスのインスタンス消去" << endl;
}
void Car::setSpeed(double speed){
    m_speed = speed;
}
```

# コンストラクタとデストラクタ

- car.cpp (Sample401)

```
double Car::getSpeed() {  
    return m_speed;  
}  
double Car::getMigration() {  
    return m_migration;  
}  
void Car::drive(double hour){  
    cout << “時速” << m_speed << “kmで” <<  
        hour << “時間走行” << endl;  
    cout << m_speed * hour << “km移動した” << endl;  
    m_migration += m_speed * hour;  
}
```

# コンストラクタとデストラクタ

- main.cpp (Sample401)

```
#include "car.h"
#include <iostream>
using namespace std;

int main() {
    Car kuruma;
    kuruma.setSpeed(40);
    kuruma.drive(1.5);
    kuruma.setSpeed(60);
    kuruma.drive(2.0);
    cout << "総走行距離:" << kuruma.getMigration()
          << "km" << endl;
    return 0;
}
```



# コンストラクタとデストラクタ

- コンパイルの仕方

コマンドプロンプトで次のコマンドを入力する

```
cl _/EHsc _main.cpp _car.cpp
```

成功したら、main.exeを実行して結果を確認

# コンストラクタとデストラクタ

- car.h (Sample401)

```
#pragma once
class Car {
public:
    Car(); ← コンストラクタは戻り値の型が不要
    ~Car(); ← デストラクタは戻り値の型が不要
    void setSpeed(double speed);
    double getSpeed();
    double getMigration();
    void drive(double hour);
private:
    double m_speed;
    double m_migration;
};
```

# コンストラクタとデストラクタ

- car.h (Sample401)

```
#pragma once
class Car {
public:
    Car();
    ~Car();
    void setSpeed(double speed); ← セッター
    double getSpeed(); ←
    double getMigration(); ← ゲッター
    void drive(double hour);
private:
    double m_speed;
    double m_migration;
};
```

# コンストラクタとデストラクタ

- car.cpp (Sample401)

```
#include "car.h"  
#include <iostream>  
using namespace std;
```

メンバ変数の初期値を設定(初期化子リスト)

```
Car::Car() : m_speed(0), m_migration(0) {  
    cout << "Carクラスのインスタンス生成" << endl;  
}  
Car::~~Car() {  
    cout << "Carクラスのインスタンス消去" << endl;  
}  
void Car::setSpeed(double speed){  
    m_speed = speed;  
}
```

# コンストラクタとデストラクタ

- main.cpp (Sample401)

```
#include "car.h"
#include <iostream>
using namespace std;

int main() {
    Car kuruma; ← コンストラクタの実行(インスタンス生成)
    kuruma.setSpeed(40);
    kuruma.drive(1.5);
    kuruma.drive(2.0);
    cout << "総走行距離:" << kuruma.getMigration()
         << "km" << endl;
    return 0; ← デストラクタの実行(インスタンス破棄)
}
```

# コンストラクタとデストラクタ

## • main.cpp (Sample401)

```
#include "car.h"  
#include <iostream>  
using namespace std
```

Carクラスのインスタンスkurumaが生成された  
時点で自動的にコンストラクタCar()が実行

```
int main() {
```

Car kuruma; ← コンストラクタの実行(インスタンス生成)

```
    kuruma.setSpeed(40);
```

```
    kuruma.drive(1.5);
```

```
    kuruma.drive(2.0);
```

```
    cout << "総走行距離:" << kuruma.getMigration()  
          << "km" << endl;
```

```
    return 0;
```

```
} ← デストラクタの実行(インスタンス破棄)
```

# コンストラクタとデストラクタ

## • main.cpp (Sample401)

```
#include "car.h"  
#include <iostream>  
using namespace std;
```

```
int main() {  
    Car kuruma;  
    kuruma.set  
    kuruma.dri  
    kuruma.dri  
    cout << "総  
        << "km" << endl;  
    return 0;  
} ← デストラクタの実行(インスタンス破棄)
```

**return**文は関数の処理を終わらせる命令

インスタンスkurumaは、main関数の中で生成されたので、**main関数が終了した時点で**

**kuruma**インスタンスが消去され、  
**デストラクタ~Car()**が自動的に実行される

# コンストラクタとデストラクタ

- まとめ

- **コンストラクタ**

- インスタンスが生成される際に必ず実行される関数  
関数名はクラス名と同じになる  
メンバ変数の初期化を行うことができる

- **デストラクタ**

- インスタンスが消去される際に必ず実行される関数  
関数名はクラス名の前に「～(チルダ)」がつく

ただし、どちらも省略可能で、なくてもよい場合もある