Geant4の基本構造と必須クラス

Geant4 10.3.P3準拠

Geant4　HEP/Space/Medicine 講習会資料

大学共同利用機関法人
高エネルギー加速器研究機構

本資料に関する注意

- 本資料の知的所有権は、高エネルギー加速器研究機構およびGeant4 collaborationが有します
- 以下のすべての条件を満たす場合に限り無料で利用することを許諾します
  - 学校、大学、公的研究機関等における教育および非軍事目的の研究開発のための利用であること
    - Geant4の開発者はいかなる軍事関連目的へのGeant4の利用を拒否します
  - このページを含むすべてのページをオリジナルのまま利用すること
    - 一部を抜き出して配布したり利用してはいけません
  - 誤字や間違いと疑われる点があれば報告する義務を負うこと
- 商業的な目的での利用、出版、電子ファイルの公開は許可なく行えません
- 本資料の最新版は以下からダウンロード可能です
  - http://geant4.kek.jp/lecture/
- 本資料に関する問い合わせ先は以下です
  - Email: lecture-feedback@geant4.kek.jp

目次

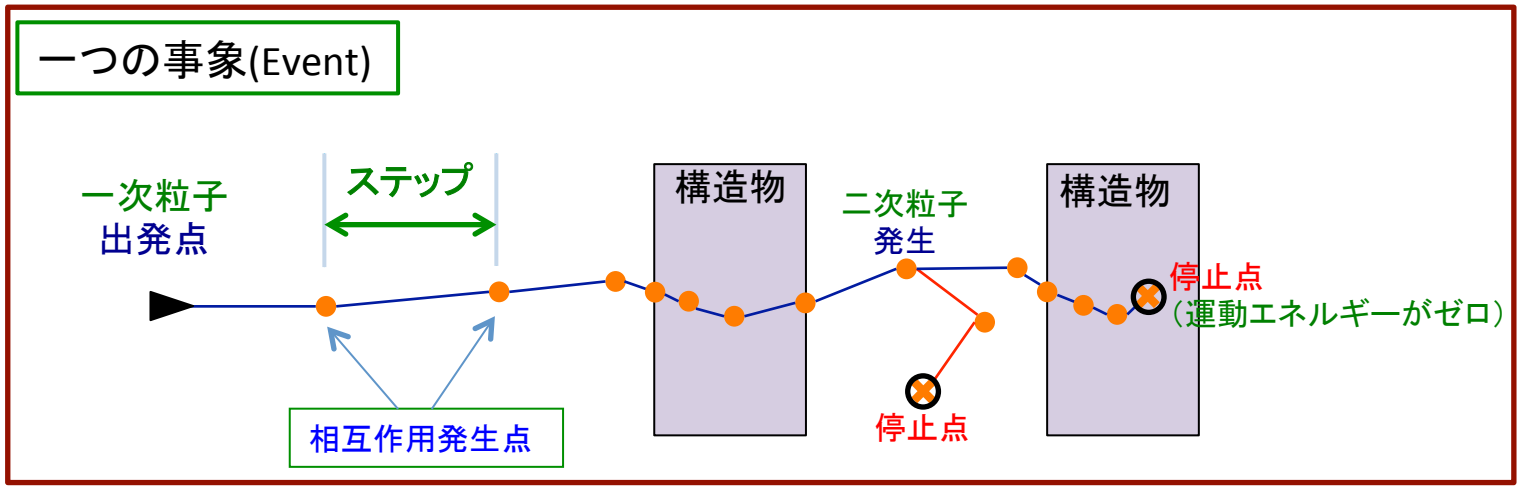
- 基本的なオブジェクト
- Geant4のプログラム構造
- ユーザクラス実装の予備知識
- Geant4アプリケーション作成の第一歩

基本的なオブジェクト

Geant4のシミュレーション手法

前の講義でシミュレーションは以下のように進むことを述べた

- 与えられた粒子進路(Track)に沿って、ステップ(Step)単位で粒子輸送が行われる
  - 相互作用の発生、構造物の境界面を通過ごとにステップが切られる
  - 従って、ステップ長さはGeant4が自動的に決定（*ユーザがコントロールすることも可能*）
- シミュレーションは事象(Event)単位で行われる
  - 事象内の全粒子の輸送が完了したら一つ事象のシミュレーションが完了
- ユーザはステップ情報、ターリ情報などでシミュレーションの結果を取得できる
- 通常、シミュレーション実行では一つのジョブ内で多数の事象を一括処理する



基本的なオブジェクトとは？

- Geant4とはオブジェクト指向言語C++で書かれたツール・キット
  - シミュレーションを構成する全ての「要素」はクラス（オブジェクト）で表現される
  - ユーザは要素クラスの中で、ユーザ用ユーティリティとして用意されたものを用い、自らのシミュレーションの枠組み（フレームワーク）を作成

- 従って、ユーザはGeant4が用意しているユーザ用ユーティリティ・クラスおよびユーザがよく使うクラスの意味を知っておくことが必要

- 以下ではユーザが知っていいるべき基本的なクラスの意味と機能をまとめる
  - ここで説明する以外にも、必要に応じて他の講義の中で新たなクラスを解説する

- 本講義ではクラス（オブジェクト）関連の用語を無理に日本語訳をしない
  - 基本的に英語を用いる
  - 必要に応じて対応する日本語を示す

Runとは？

- 同一の条件のもとで一連の事象をシミュレーションすることを'Run'を実行するという
  - 'Run'という言葉は、粒子加速器実験で、粒子ビームの照射開始から終わりまでの一連の測定に対して名付けられたもの
- 'Run'の開始時に以下の設定が自動的に行われる
  - シミュレーションで考慮される構造物（計測機器など）の幾何情報('Geometry')の最適化
  - 構成物質と粒子の相互作用の反応断面積の計算
  - 二次粒子生成条件の設定 （ユーザが設定した値に従って）
- 一つの'Run'の実行中は、以下のシミュレーション条件をユーザが変更することは不可
  - 構造物のセットアップ
  - シミュレーションで考慮する物理過程
- 一つのRunに関連する情報は' G4Run'クラスで表現される
  - *'G4Run'*クラスは一つの'Run'でのシミュレーション結果をまとめて保持する
- '*G4RunManager*'クラスは'Run'の間の全てのシミュレーション処理を制御

  [注] G4RunManagerはユーザがフレームワーク（アプリケーション）を簡単に作れるように用意されている。ほとんどのユーザはこれを使い、main()プログラムを書くだけで自分のシミュレーションを作成できる。熟練ユーザはこれを使わずフレームワークを作成することも可能
- '*G4UserRunAction*'クラスは'Run'中にユーザがシミュレーションに介在するためのユーザフック

Eventとは？

- ‘Event‘（事象）とは初期粒子（’Primary Particles’ ：一次粒子ともいう）が構造物に入射したことで生じさせる物理現象の全てをさす
  - 前述のRunは複数のEventsの集合
- ’Primary Particles’の輸送（Tracking）が開始されることで一つの‘Event‘は始まる
- ‘Tracking‘中に生成される2/3/4….次粒子が生じる物理現象も’Event’に含まれる
- 初期粒子、2/3/4….次粒子が全て‘Tracking‘されることで一つの‘Event‘処理が完了
- ’*G4Event*’クラスは一つの‘Event‘を表現し、以下の情報を保持する
  - 一次粒子の情報及びそれらの空間始点情報 （初期情報）
  - 生成された2/3/4….次粒子のシミュレーション経過情報（Trajectoryとよぶ)のなど
- ’*G4EventManager*’クラスが一つの‘Event‘のシミュレーションを制御
- ’*G4UserEventAction*’クラスは‘Event‘処理中に必要に応じてユーザがシミュレーションに介入出来るためのユーザ・フック

Trackとは？

- Geant4は粒子輸送（'tracking'）をStep(後述）単位で進める
- Trackingは以下の条件で終了する
  - 粒子が設定されている幾何学的な全空間（ワールド）の外に出た時
  - 粒子が崩壊、非弾性散乱等で消滅した時
  - 粒子の運動エネルギーがゼロ（すなわち静止状態）
  - ユーザが強制的に粒子のtrackingを停止した時
- 'Track'はtracking中の粒子の現時点でのStep情報を保持するとともに、tacking開始時点からの現在までの総step数、経過時間、trackの長さなどの情報を保持
  - Trackは現時点での'step'情報のみを保持し、過去の'step'情報は保持しない
  - 情報の詳細と取得方法については以下のwebページ（FAQ: 4. Tracks and steps）を参照

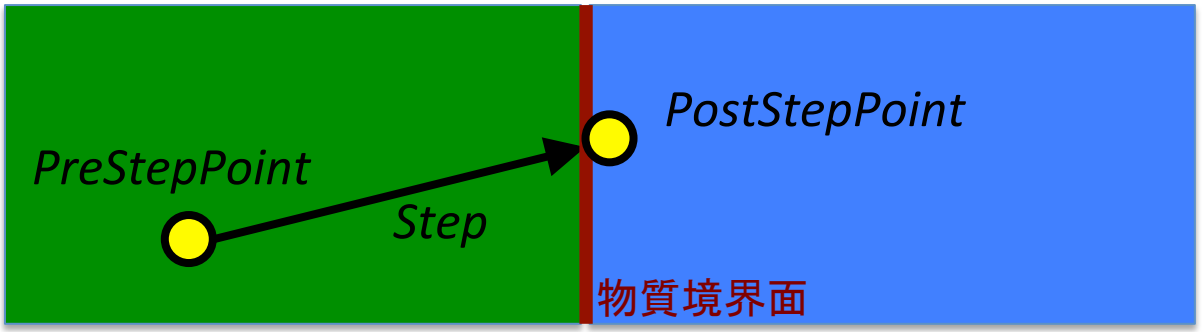
    http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/FAQ/html/index.html
- Trackの保持する情報はevent処理の終了時に破棄
  - Track情報を保持したい場合は、eventが持つ'Trajectory'に書き加える
- Trackは'*G4Track*'クラスで表現
- '*G4TrackingManager*'クラスは粒子のtrackingを制御
- '*G4UserTrackingAction*'クラスは tracking処理中に必要に応じてユーザが介入出来るためのユーザ・フック

Stepとは？

- Stepはtrackingの最小単位
  - 粒子を一つのstepだけ進めることを*stepping*とよぶ
- Stepの始点を*Pre-step Point*、終点を*Post-step point*とよび、この二つのオブジェクトでstepの前後での粒子の物理状態の変化、ジオメトリ情報などを知ることができる
- Stepは'*G4Step*'クラスで表現。Stepの始点と終点は'*G4StepPoint*'クラスで表現される
  - 情報の詳細と取得方法については以下のwebページ（FAQ: 4. Tracks and steps）を参照

    http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/FAQ/html/index.html
- '*G4SteppingManager*'クラスは粒子の一つのsteppingを制御
- '*G4UserSteppingAction*'クラスは各stepでの情報を得るためのユーザ・フック
  - このフックを用いてユーザはtracking過程の最も詳細な情報にアクセスが可能

Stepの終点が物質境界面にある場合、終点は境界面の先にある物体の中に置かれる



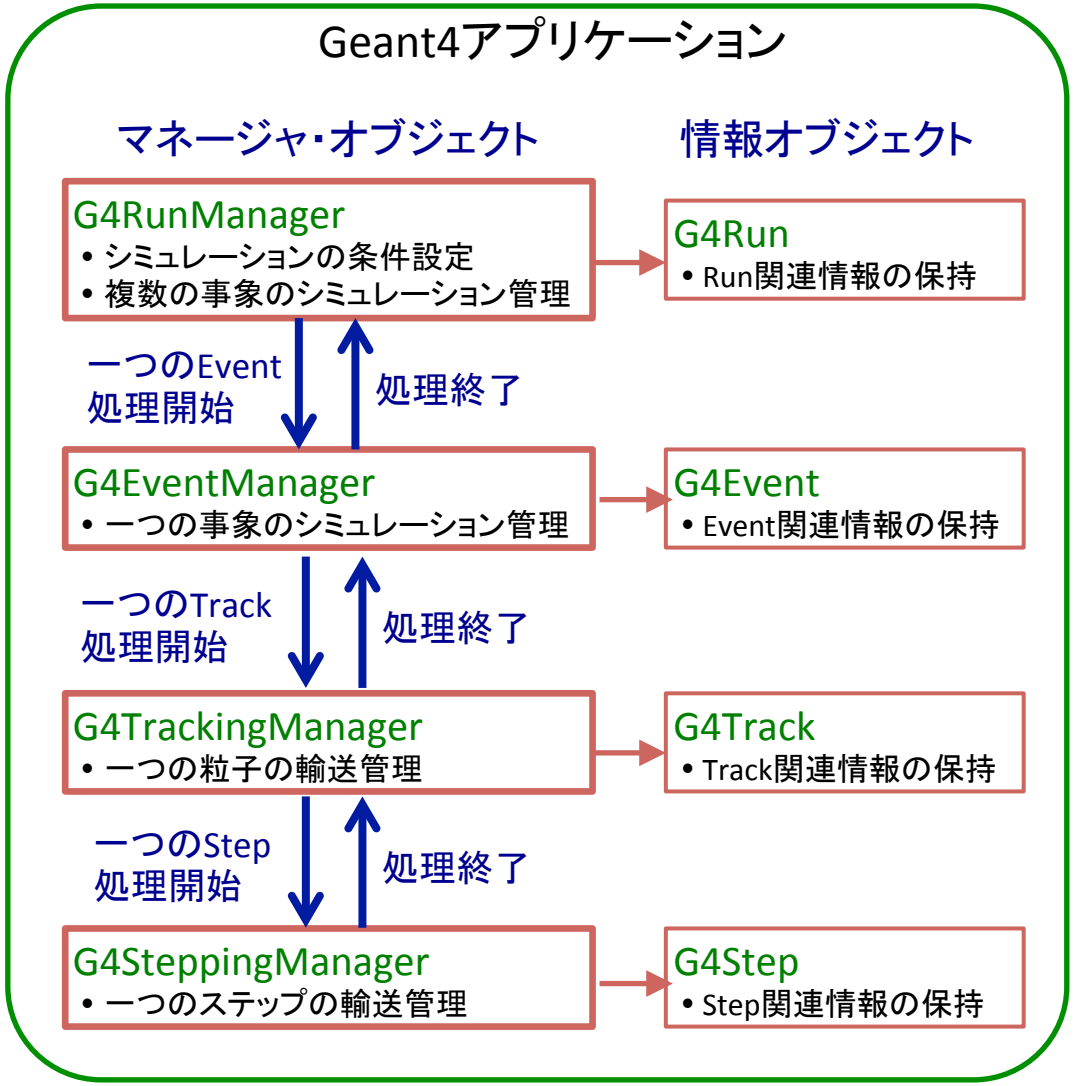
Particleとは？

- Geant4で定義されている様々な素粒子、原子核(particles)などを表現する
- 以下のようなタイプのものが定義されている
  - 通常の粒子： 電子、陽子、中性子、π粒子、μ粒子、など
  - 共鳴による短寿命な粒子： Δ、N、ρ、ω、など
  - 原子核： α、重イオン、など
  - クォーク： u, d, s, など
- これらのparticleは以下のようにグループ分けして管理されている
  - レプトン
  - ハドロン ➔ バリオン、メソン、イオン
  - ボゾン および 短寿命粒子
- それぞれのparticleは'*G4ParticleDefinition*'クラスから継承されて定義されている
  - class G4Electron : public G4ParticleDefinition
  - class G4Gamma : public G4ParticleDefinition
  - ……
- それぞれのparticleはどのような物理相互作用を起こすかの情報を保持
- 定義されている全particlesはGeant4アプリケーション起動時に自動的に生成
- 定義されているparticleは'*G4ParticleTable*'クラスを通してアクセスできる
- G4Trackは粒子タイプ情報としてG4ParticleDefinitionオブジェクトを持っている

Geant4のプログラム構造

Geant4アプリケーションの基本構造

- シミュレーションは4つの階層構造を通して自動進行
- 階層内でのシミュレーション進行の管理はマネージャ・オブジェクトが行う
- それぞれの階層で作られた情報は情報オブジェクトに保持
- ユーザは最上位のマネージャである*G4RunManager*に必要な初期情報を渡すだけでシミュレーションを実行できる
  - 次のスライド参照



ユーザは何をしなければならないか？

- ツールキットが提供するC++クラスを使って、次の3つの初期情報を*G4RunManager*に通知するだけでシミュレーションを実行できる
  - 1） シミュレーション空間に存在する構造物の情報
    - 構造物を構成する物体、物質の定義
  - 2） 考慮する物理相互作用
    - 考慮すべき粒子、物理相互作用／モデルの定義
    - 二次粒子生成閾値の設定
  - 3） 事象の初期条件
    - 一次粒子（一つの事象をシミュレーション開始する際に、初期に存在する粒子）の初期条件（場所、エネルギー、等）の定義

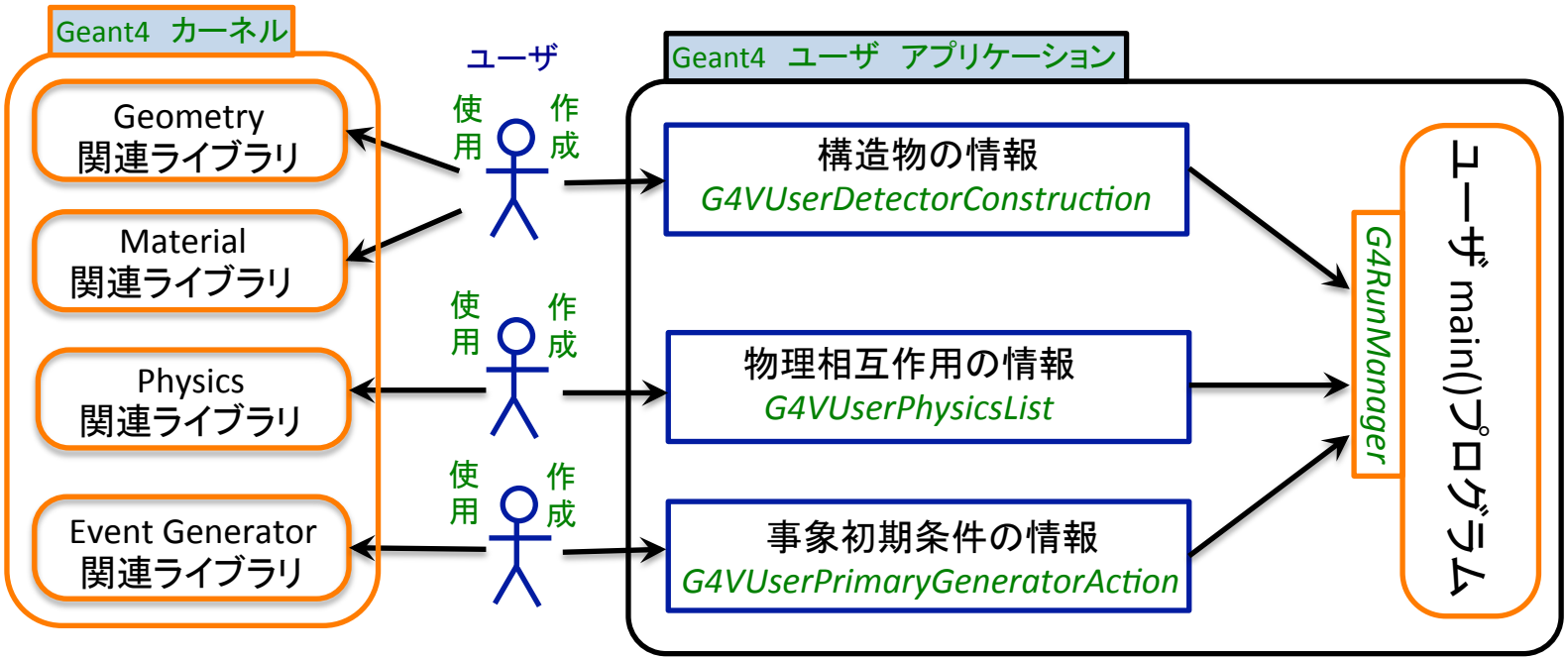
    [注] 単に定義した構造物をGeant4で描画させたいだけであれば（事象シミュレーションを実行はしないこと）、この定義は省略することが可
- 上記初期情報のG4RunManagerへの通知はユーザアプリケーションのmain()関数の中で行う
  - すなわち、ユーザはmain()を自ら書く必要がある
    - ← 実際には、Geant4が提供する例題(サンプルプログラム)からそのmain()をコピーし、必要に応じてそれに変更を加えるだけでOK

ユーザクラスとは？

- ユーザクラスとはユーザがシミュレーション初期情報をGeant4カーネルに伝える時に使用する基底クラスの総称
- ユーザがアプリケーションを作成するには、このクラスの基本的な使い方さえ理解すればよい
- ユーザは以下の3つのユーザクラスからオブジェクトを生成して*G4RunManager*に渡せば、シミュレーションが開始可能:

1） 構造物 ➔ *G4VUserDetectorConstruction*
2） 粒子と物質の相互作用 ➔ *G4VUserPhysicsList*
3） 事象初期条件 ➔ *G4VUserPrimaryGeneratorAction*

[注] この情報を渡さなくてもアプリケーションは動作するが、事象のシミュレーションは開始されない



ユーザクラスの分類

■ ユーザクラスはその機能で以下の二つに分類

1. 初期化必須クラス(Mandatory Initialization Classes)

[機能] シミュレーションの初期化時に必要となる情報を伝達

- G4VUserDetectorConstruction [必須作成クラス]
  - 構造体情報
- G4VUserPhysicsList [必須作成クラス]
  - シミュレーションで考慮したい粒子、およびその粒子と物質の相互作用

2. ユーザアクション•クラス(User Action Classes)

[機能] 事象シミュレーションを開始するあたり必要となる情報を伝達(全てが必須ではない)

- G4VUserPrimaryGeneratorAction [必須作成クラス] [注] *構造体の描画だけなら不要*
  - 事象の初期条件
- G4UserRunAction
  - 一つの'Run'処理にユーザが介入できるユーザフック
- G4UserEventAction
  - 一つの'Event'処理にユーザが介入できるユーザフック
- G4UserTrackingAction
  - 一つの'Track'処理にユーザが介入できるユーザフック
- G4UserSteppingAction
  - 一つの' Step'処理にユーザが介入できるユーザフック
- G4UserStackingAction
  - Track処理順序をユーザが制御するためのユーザフック

シミュレーション結果の取得

- Geant4は前述の3つの情報が*G4RunManager*に渡されれば、あとは、自動的に全事象のの粒子輸送(tracking)を実行
  - デフォルトは最小情報のみを実行端末に書き出した後、シミュレーションを終了
  - ユーザが望むシミュレーション結果情報の取得には、独自のコード(通常C++) を書くか、あるいは標準的に用意されている対話型コマンド(UIコマンド)を使用することが必要

- 結果情報の取得には以下の*二通りの方法*が用意されている
  - ユーザフック(*G4UserTrackihngAction, G4UserSteppingAction*等)の使用
    - フックからシミュレーション結果のあらゆる情報を入手可能
    - 手法は単純であるが、ユーザがコードを書くことが必要
    - 個々の粒子が輸送される経過を知りたい場合に使う （例：HEPの測定器シミュレーション）
  - スコアラー(scorer、scoringともよぶ)を使う。
    - スコアラー関連コマンド(UIコマンド)を用いて必要情報をファイルに書き出す
    - 有感物体を指定し、必要な物理量をスコアリングするC++コードを書く
    - 複数事象の積算で得られる情報を知りたい場合に使う （例：医療での線量計算）

  [注] 結果情報取得の具体的手法は「スコアリング」の講義で詳しく述べる

ユーザ支援機能

- ユーザが前述の基本情報を記述する際に使用できる、便利なC++クラスが用意されている。また、それ以外にも、シミュレーション作成にあたって、ユーザを支援する様々なクラスが多く用意されている

- 例:
  - ユーザが定義した構造物の可視化
  - Trackingされた粒子のトラジェクトリ可視化
  - シミュレーション処理の各段階でユーザが介入できる多種のコマンド
  - (グラフィカル)ユーザ・インターフェイス
  - マクロ作成
  - その他

[注] これらの機能については後のコースで内容を詳しく述べる

ユーザクラス実装の予備知識

C++の継承を使いこなす

- Geant4は完成したシミュレーション実行コード（アプリケーション）ではなく、C++で記述されたツールキットである
- ユーザはアプリケーションを作成するにあたり、ツールキットが提供する仮想クラスを使い（継承して）必要なクラスを作成したのち、それらを使ってアプリケーションのmain()プログラムを書く
  - Geant4はオブジェクト指向で設計され実装されているので、*仮想クラス*、*基底クラス*、*継承*、*オーバーロード*などのC++基礎概念を理解していることが必要
  - たとえば、先に述べた初期化必須クラスは仮想クラス（次のスライド参照）として提供されているので、それを継承して自分のクラスを実装することになる
  - その他、目的に応じてユーザが作成するクラスも、継承すべき基底クラスが用意されているものが多い

[注]

  - 以上の話を聞くと、アプリケーションを作るのは大変という印象を持つかもしれないが、本講座でGeant4の基礎を学べば実は簡単だということが分かるだあろう
  - Hands-onの例題のコードを読んで実行してみることが最短距離である

継承の鍵は仮想(virtual)

- クラスの*仮想関数*と*純粋仮想関数*
  - virtual func(){ ; } 仮想関数 （実装されている）
  - virtual func() = 0; 純粋仮想関数 （実装がない）
- クラス定義で一つでも”*純粋仮想関数*”が宣言されていれば、それは抽象クラスである
- 抽象クラスはそれから直接オブジェクトを生成することはできない
  - 純粋仮想関数を実装し、派生クラスを定義することでオブジェクトが生成可能
- 先に述べた初期化必須クラスは抽象クラスで記述されている
  - 例えば、geometryには”G4VUserDetectorConstruction”クラスがある
    （V はvirtualを意味 - 命名規則として抽象クラスにはV はを付加）
  - その中で定義されている下の関数は”純粋仮想関数”

```
virtual G4VPhysicalVolume* Construct() = 0;
```

- ユーザが構造物の形状を定義するときこの”G4VUserDetectorConstruction”クラスを使う。しかし、抽象クラスで定義されているため、ユーザは必ず派生クラスを定義し、その中で*Construct()*関数を実装しなければならない

継承の鍵は仮想(virtual) ― 続き

- ユーザ•アクションクラスは抽象クラスとしては定義されていない
  - 例えば、G4UserSteppingActionクラスの次の関数

```
virtual void UserSteppingAction(const G4Step*) { ; }
```

- 従って、ユーザ•アクションクラスからはオブジェクト(インスタンス)を直接作ることが可能である
  - 例えば、次の様なダミーのオブジェクトを作れる

```
G4UserSteppingAction dummySteppingAction;
```

[注] 先に述べた抽象クラスは、直接オブジェクトは作れないが、以下のようにポインターの宣言はできる:

```
G4VUserDetectorConstruction* detector;
```

Geant4の入出力ストリーム

- *G4cout*と*G4cerr*はGeant4が提供する出力ストリームである。また、*G4endl*も提供されている。

```
G4cout << “Hello Geant4!” << G4endl;
```

- Geant4のGUIのいくつかは上記の出力ストリームを通して出力値をバッファリングする。そしてその値は画面表示に用いられたり、あるいは保存や編集に用いられる
  - 以上の理由から、ユーザはC++クラス・ライブラリの”std::cout”及び”std::cerr”をGeant4アプリケーションでは使うべきでない
- G4cinはGeant4が提供する入力ストリームである。Geant4アプリケーションではC++クラス・ライブラリの”std::cin”ではなく、G4cinを使うことをすすめる

Geant4が用いる単位系

- Geant4が内部的に使っている単位系は、ユーザのみならずGeant4 toolkit自体のソースコードからもあらわに見えないように設計されている
- そのかわりに、ユーザが単位を持つ変数を定義する時は、必ず単位を明記する

```
Radius = 10. 0 * cm;
kineticE = 1.0 * GeV;
momentum = 5.0 * GeV;
```

← 単位を*かけ算*で付する

[注] 運動量は内部単位（次のスライド参照）でMeV/c、
従ってGeV/cで運動量を表現したいならGeVをかける

- また、単位を持つ変数から物理値を取るにも、必ず単位を明記する必要がある

```
G4cout << eDep / MeV << " [Mev]" << G4endl;
```

← 単位を*わり算*で付する

- 上の例で示された様な良く用いる単位表示変数(cm, GeV, MeV等）は、Geant4がデフォールトで定義しており、ユーザはそれを単に使うだけでよい。ただし、以下の宣言を".cc"ファイル中にしておくこと

```
#include "G4SystemOfUnits.hh"
```

← Geant4 9.6以前では'globals.hh'の宣言で良かったがV10.0以降で変更された

- また、ユーザが独自の単位を定義する事も可能
- この様に単位系を扱うことで、プログラムコードは読みやすくなり、しかも物理量を外部に書き出す、或は外部から読み込む際の単位の混乱は避けられる

Geant4が用いる単位系　ー　続き

- ユーザが故意に単位を明示せずに物理変数を定義すると(これは絶対に避けるべきであるが)、その物理量は以下のGeant4内部単位系を持つものとして処理される:

  長さ: mm　　時間: ns　　エネルギー: Mev　　温度: Kelvin

  物質量: mole　　光度: candela　　角度: radian　　立体角: steradian

  電荷: positron charge

- Geant4で標準的に使える単位の一覧を見るには以下のようにする
  - UI コマンド:　　Idle> /units/list
  - プログラムコード:　　G4UnitDefinition::PrintUnitsTable();

- 単位を持つ変数から物理値を取る時に、その値に応じて最適の単位を自動的にえらぶことができる

```
#include “G4UnitsTable.hh”
G4cout << G4BestUnit(eDep, “Energy”) << G4endl;
```

“Length”, Time“, “Energy”,”Volume”, “Mass”,….
単位一覧の中のカテゴリー名

Geant4で使える物理定数

- Geant4は標準的な物理定数を提供している。これはtoolkit内で使われているとともに、ユーザが自らのアプリケーションのコード内で使える

```
G4double sectorAngle = twopi/nSector;
G4double density = universe_mean_density;
```

[注] 上の例でtwopi、universe_mean_densityはGeant4が提供する物理定数

twopi = 2.0*3.14159265358979323846;

universe_mean_density = $1.e^{-25}$*g/cm^3;

- 提供されている定数の例

pi, twopi, halfpi 、pi2、c_light、c_squared、h_Planck、hbar_Planck、hbarc、hbarc_squared、electron_charge、e_squared、electron_mass_c2、proton_mass_c2、neutron_mass_c2、elm_coupling、fine_structure_const、classic_electr_radius、electron_Compton_length、等

- Geant4で標準的に備えている物理定数の一覧は以下のソースにある

Geant4/externals/clhep/include/CLHEP/Units/PhysicalConstants.h

- ユーザが標準的な物理定数を使う場合、以下の宣言を".cc"ファイル中にする

#include "G4PhysicalConstants.hh" ← Geant4 9.6以前では'globals.hh'の宣言で良かったがV10.0以降で変更された

Geant4アプリケーション作成の第一歩

ユーザアプリケーションのmain関数の作成

■ 基本的な手続きは以下のようになる:

1） main()の作成

2） G4RunManagerオブジェクトを作成

3） ジオメトリ（*G4VUserDetectorConstruction*）記述オブジェクトをG4RunManagerに直接登録

4） 物理相互作用（*G4VUserPhysicsList*）記述オブジェクトをG4RunManagerに直接登録

5） 事象初期条件（*G4VUserPrimaryGeneratorAction*）記述オブジェクトをG4RunManagerに登録

- G4VUserActionInitializationを通してG4RunManagerに登録 （Geant4.10以降）
- G4RunManagerに直接登録 （Geant4.10より以前、*ただしGeant4.10以降でもOK*）

```
//---------------------------------------------------------------------
int main( int argc, char** argv )
//---------------------------------------------------------------------
{
// Construct the default run manager
   G4RunManager * runManager = new G4RunManager;

// Set up mandatory user initialization: Geometry
   runManager->SetUserInitialization( new Geometry );

// Set up mandatory user initialization: Physics-List
   runManager->SetUserInitialization( new FTFP_BERT );

// Set up user initialization: User Actions
   runManager->SetUserInitialization( new UserActionInitialization );

// Initialize G4 kernel
   runManager->Initialize();

   ……
}
```

三つの
必須作成クラス

main()の詳細は
Hands-Onをとおして学ぶ