1. **Jakie elementy dziedziczy klasa pochodna po klasie bazowej?**

W przypadku publicznego dziedziczenia klasa pochodna dziedziczy wszystkie składowe i metody klasy bazowej. Elementy prywatne są dostępne ale tylko za pośrednictwem publicznych i chronionych elementów klasy bazowej, choć prywatne skłądowe prywatne też są dziedziczone ale nie wolno się do nich bezpośrednio odnosić, poza kontruktorami destruktorami opratorami przypisania i funcjami zaprzyjaznionymi.

Reasumując:   
Klasa pochodna dziedziczy dane składowej typu bazowego, implemetnacje klasy bazowej, metody typu bazowego i interfejs klasy bazowej.

1. **Jakich elementów nie dzidziczy klasa pochodna po klasie bazowej?**

Konstruktory, destruktory klasy bazowej, operatory przypisania oraz funkcje i klasy zaprzyjaźnione

1. **Załóżmy że typ zwracany przez funkcję *baseDMA::operator=()* został zdefiniowany jako void zamiast baseDMA &. Jakie skutki ma taka definicja? Co się stanie jeśli zwracany typ zostanie zdefiniowany jak baseDMA zamiast baseDMA&?**   
     
   Jeśli void, to nie zwróci on niczego.   
   Inny typ niż baseDMA spowoduje że operator nie będzie też mógł zwrócić obiektu danej klasy.   
   Stosuje się go aby przypisać jeden obiekt do drugo, jeśli funkcja nie jest zaprzyjaźniona a tylko po prostu „void” to będzie poza klasą a więc będzie miała dostęp do klasy tylko do składowych publicznych, a więc nie będzie mogła operować na danych prywatnych.   
     
     
   BaseDMA vs baseDMA& - różnica w refencji. Metoda zwraca wartość (jeśli bez &) albo adres do obiektu przez referencję właśnie. Różnica jest przede wszystkim wydajnościowa bo w drugim przypadku dostajemy oryginalny obiekt o tyle w pierwszym musimy zrobić kopie obiektu zmieniać go a następnie zwrócić już obiekt zupełnie inny niż go przekazaliśmy.   
     
   Dodatkowo void nie można przypisywać w sposób łańcuchowy tj.  
   klasaBazowa magazine (…, …, …,)  
   klasaBazowa gift1, gift2, gift3;  
   gift1 = magazine; // OK  
   gift2 = gift3 = magazine; //Instrukcja już nie będzie prawidłowa
2. **W jakiej kolejności wywoływane są konstruktory i destruktory w momencie tworzenia i usuwania obiektów klasy pochodnej?**  
   Kiedy program tworzy obiekt klasy pochodnej, najpierw musi utworzyć klasy bazowej, a więc najpierw **konstruktor klasy bazowej**. Do tego potrzebna jest lista inicjalizacyjna, jeśli ją pominiemy, to program wywoła konstruktora domyślnego klasy bazowej.(697 s)   
     
   **Destruktory**: Najpierw wywoływany jest destruktor klasy pochodnej a potem bazowej ale to tyczy się destruktora typu wirtualnego!  
   Jeśli destruktor nie jest wirtualny to wywoływany jest najpierw destruktor typu wskaźnika, więc bazowy raczej, chyba że wskażemy inaczej. (720 s)   
     
   Konstruktor klasy znajdujacy się na szczycie hierachii dziedziczenia pierwszy zostanie wywołany, destruktory są wywoływane w kolejności odwrotnej.
3. **Czy trzeba w klasie pochodnej definiować konstruktor, jeśli nie ma w niej żadnych nowych składowych?**  
     
   Tak, ponieważ bez deklaracji konstruktora klasy pochodnej nie ma listy inicjalizacyjnej na konstruktor klasy bazowej, a gdy nie ma takiej listy inicjalizacyjnej kompilator wywołuje domyślny konstruktor klasy bazowej, nawet jeśli zadeklarowane są konstruktory w klasie bazowej.
4. **Załóżmy że w klasie bazowej i w klasie pochodnej zdefiniowana jest metoda o takiej samej nazwie. Program wywołuje tę metodę dla obiektu klasy pochodnej. Która wersja metody zostanie użyta.**   
     
   Wywoływana jest metoda albo na podstawie *typu referencji* albo na podstawie *typu obiektu.*   
   Jeśli *typu obiektu* to wywoływana jest funkcja której typ dotyczy klasy w tym wypadku zostanie wykonana metoda *klasy pochodnej.*  
   Jeśli *typ referencji,* np. typ obiektu jest *klasy bazowej* ale referencja przypisana jest do niego z *klasy pochodnej* to wtedy są dwie mozliwosci:  
   Gdy nie ma kluczowego **virtual** w *metodzie bazowej* i *pochodnej* to wtedy wykona się metoda na podstawie *typu obiektu* to wykona się wg *typu klasy bazowej,*  
   ale jeśli mamy funkcje **virtual** to wykona się na podstawie *typu referencji* czyli metoda z *klasy pochodnej.*
5. **Kiedy powinno się definiować operator przypisania w klasie pochodnej?**  
     
   Wtedy kiedy chcesz przypisać jeden obiekt do drugiego tej samej klasy, a na pewno wtedy kiedy chcemy przypisać obiekt klasy bazowej do obiektu klasy pochodnej i zdefiniowany jest konstruktor konwersji tj. **klasaPochodna(const klasaBazowa &)**;   
   w takiej sytuacji jedynie wskaźnik lub referncja mogą wywołać metody klasy bazowej!  
     
   Jeśli konstruktor klasy używa operatora **new** lub new [] to wtedy należy zdefiniować jawny operator przypisania!   
   Ogólnie powinno się definiować operator przypisania wtedy kiedy nie jestnieje poprawne przypisanie domyślne.
6. **Czy można przypisać adres obiektu klasy pochodnej do wkaźnika na klasę bazową? Czy można przypisać adres obiektu klasy bazowej do wskaźnika na klasę pochodną?**  
     
   Obiekt *klasy pochodnej* **możemy** przypisać na wskaźnik i referencję *klasy bazowej,*  
   Obiekt *klasy bazowej* **nie możemy** przypisać na wskaźnik i referencje *klasy bazowej*!  
     
    **KlasaPochodna** rplayer(..., ..., ...);

**KlasaBazowa** &rt = rplayer; => rt.**MetodaKlasyBazowej**() ;referencja

**KlasaBazowa** \*pt = &rplayer; => pt->**MetodaKlasyBazowej**() wskaźnik

Jedynie wkaźniki lub referencję klasy bazowej mogą wywołać metody *klasy bazowej*!

**KlasaBazowa** player(..., ..., ...);

**KlasaPochodna** &rr = player; //NIEDOZWOLONE

**KlasaPochodna** \*pr = &player; //NIEDOZWOLONE  
  
Adres obiektu klasy bazowej można przypisać wskaźnikowi na klasę pochodną to rzutowanie w dół (downcasting)

Rzutowanie w dół jest możliwe przez jawne rzutowanie typu, a korzystanie z takiego wskaźnika niekoniecznie jest bezpieczne.

1. **Czy można przypisać obiekt klasy pochodnej do obiektu klasy bazowej? Czy można przypisać obiekt klasy bazowej do obiektu klasy pochodnej.**  
   Obiekt klasy pochodnej do obiektu klasy bazowej można przypisać ale gubimy wtedy składowe dodatkowe dla klasy pochodnej, ale nie można tego zrobić bez jawnego rzutowania typów.  
   Obiekt klasy bazowej do obiektu klasy pochodnej można przypisać najlepiej wykorzystać do tego rzutowanie w górę (up casting)
2. **Załóżmy, że zdefiniowałeś funkcję przyjmującą jako argument referencje do obiektu klasy bazowej. Dlaczego funkcja ta może używać jako argumentu także obiektu klasy pochodnej?**   
     
   Jeżeli ta funkcja o tej samej nazwie została zdefiniowana w klasie pochodnej i odwołuje się do funkcji w klasie bazowej to nie ma problemu.  
   Ale najważniejsze jest rzutowanie typu *klasy pochodnej* na *klasę bazową*, co pozwala na skorzystanie z funkcji *klasy bazowej* przez obiekt *klasy pochodnej.*   
   Istotny jest paremetr **const NazwaKlasyPochodnej &** rzutowany jest argument na typ **const NazwaKlasyBazowej &**. (str 748-749)
3. **Załóżmy że zdefiniowałeś funkcję przyjmującą jako argument obiekt klasy bazowej, który przekazywany jest przez wartość. Dlaczego funkcja ta może używać jako argumentu także obiektu klasy pochodnej.**   
     
   ~~Tak jest to możliwe ale jedynie wtedy kiedy funkcja jest metodą wirtualną danej klasy.~~Przekazywanie obiektu przez wartość powoduje wywołanie konstruktora kopiującego. Ponieważ parametr formalny jest obiektem klasy bazowej, wywołany jest konstruktor kopiujący klasy bazowej. Argumentem konstruktora kopiującego jest referencja do klasy bazowej, która może się także odnosić do przekazanego w postaci argumentu obiektu pochodnego. Efekt kocowy jest taki, że zostaje utworzony nowy obiekt klasy bazowej, którego składowe odpowiadają części bazowej obiektu pochodnego.
4. **Dlaczego zwykle lepiej przekazać obiekty przez referencje niż przez wartość?**  
     
   Przekazując przez wartość kompilator automatycznie tworzy dodatkowy obiekt który jest kopią tego przekazywanego i jednocześnie zajmuje to pamięć.  
   Przez referencję nie trzeba tworzyć drugiego obiektu, ponieważ przekazywany jest adres obiektu który wywołuje daną metodę przez co takie wywoływanie jest szybsze.  
     
   Przekazywanie przez referencje a nie przez wartośc umożliwia funkcji skorzystanie z funkcji writualnych.   
   Przekazywanie przez wartość daje taki plus że nie narusza on danych przekazywanych, choć ten sam efekt można uzyskać dając const przy referencji.
5. **Załóżmy, że *Organizacja* jest klasą bazową dla klasy pochodnej *OrganizacjaPubliczna*. Załóżmy też, że dla każdej z tych klas zdefiniowana jest funkcja składowa *head()* , że *ph* to wskaźnik na typ *Organizacja* oraz że do *ph* przypisany jest adres obiektu typu *OrganizacjaPubliczna.* Jak zostanie zinterpretowane wywołanie *ph→head(),* jeśli w klasie bazowej funkcja *head()* zdefiniowana jest jako:**   
   a) Zwykła metoda niewirtualna.  
   b) Metoda wirtualna  
     
   mamy odpowiednio tak:  
   *OrganizacjaPubliczna cos(..,..,…);*  
   *Organizacja \*ph = &cos;  
   ph →head() ???*   
     
   a) Wywołana zostanie metoda klasy bazowej  
   b) Wywołana zostanie metoda klasy pochodnej. (Przykład list. 13.10 z tablicą Brass)
6. **Jakie błędy, jeśli w ogóle, znajdują się w poniższym kodzie?**

class Kitchen

{

private:

double kit\_sq\_ft;

public:

Kitchen() {kit\_sq\_ft = 0.0; }

virtual double area() const { return kit\_sq\_ft \* kit\_sq\_ft;}

};

class House: public Kitchen

{

private:

double all\_sq\_ft;

public:

House() {all\_sq\_ft += kit\_sq\_ft; }

double area (const char \*s) const {cout << s; return all\_sq\_ft;}

}

1. Jest czymś… powinno być odwrotnie, to Kuchnia powinna dziedziczyć po House ze względu na to że to Kuchnia jest częścią domu a nie dom częścią Kuchni.

2. area() w klasie pochodnej House nie jest virtual tak jak w klasie bazowej Kitchen definicja metody area() w klasie House przykrywa wersję funkcji area() zdefniowaną w klasie kitchen, ponieważ te dwie metody posiadają różne sygnatury.

3. area z klasy bazowej Kichten nie ma danych wejściowych więc wynik takiego działania będzie zawsze zero!  
4. Nazewnictwo kit\_sq\_ft – to już jest pole powierzchni więc w klasie bazowej area wychodzi do ^4 powierzchnia :)

5. Klasa pochodna House nie ma dostępu do prywatnych składowych klasy bazowej Kitchen więc nie można operować z poziomu konstruktora House() na kit\_sq\_ft.

6. all\_sq\_ft w klasie pochodnej nie ma metody która by w jakikolwiek sposób operowała na tej danej, więc jest to zmienna w zasadzie pusta.