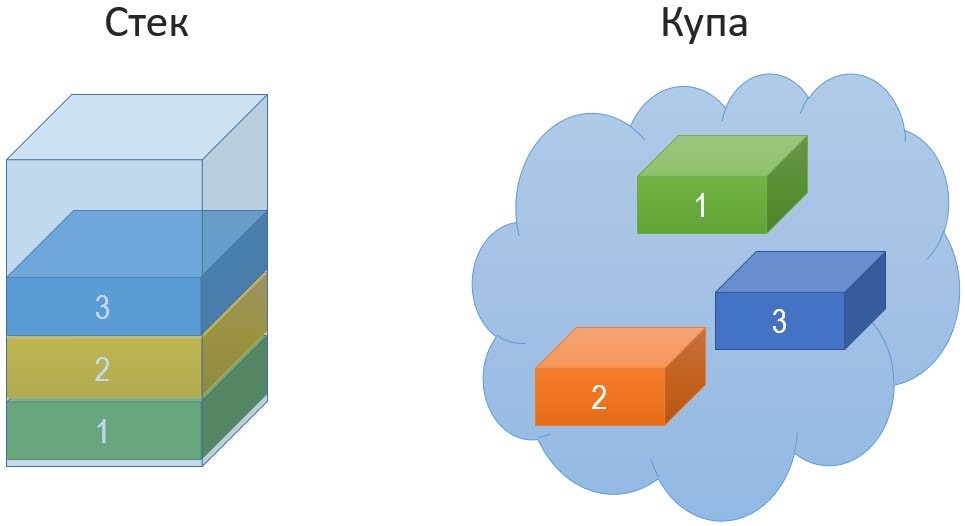
#### Розміщення екземплярів у пам'яті

Під час роботи програми її дані мають бути розташовані у пам'яті. Обсяг пам'яті, необхідний для розміщення об'єкта, залежить від конкретного типу та його елементів даних. Застосунки можуть розташовувати свої дані в оперативній пам'яті двох видів: у *стеку (stack)* та в *купі (heap)*. Як правило, програміст не має потреби безпосередньо працювати зі стеком та купою, оскільки розподілом пам'яті керують внутрішні механізми C# та .NET. Замість цього програміст у програмі оперує об'єктами, даючи вказівки створити об'єкт чи видалити його. А виконання цих інструкцій автоматично запускає внутрішній системний механізм для їх реалізації. Проте, знання основ функціонування стеку та купи дозволяє краще зрозуміти, що відбувається під час роботи програми, тому розглянемо їх детальніше.

*Стек* – це пам'ять, робота якої організована за принципом "останнім зайшов – першим вийшов". Стек можна уявити як коробку, в яку одна на одну складають папки з документами (рис. 2.6). Витягнути папки з неї можна тільки починаючи з кінця, причому одна за одною, в зворотному порядку.

Рис. 2.6. Організація стеку та купи. 1,2,3 – структури даних

У стеку зберігаються значення змінних деяких типів, параметри методів та стан поточного середовища виконання програми. Дані в стек можна додавати тільки на його вершину.

*Купа* – це пам'ять, ділянки якої використовують для розміщення різних об'єктів. На відміну від стеку, дані можуть зберігатися в купі у будь-якому порядку (рис. 2.6). Порядок звертання до даних також довільний.

Екземпляри користувацьких типів (класів), які створює програма, розміщуються у купі. При цьому для них виділяється потрібна ділянка пам'яті. Але самостійно вивільняти цю пам'ять у програмі на C# не потрібно (на відміну від C та C++, де вивільнення об'єктів мав ретельно відслідковувати програміст для запобігання втрат пам'яті). Цю роботу автоматично виконує компонент .NET Framework, який називають "збирачем сміття" (garbage collector)[[1]](#footnote-1). Збирач сміття виявляє об'єкти у купі, які код більше не використовує, і знищує їх, вивільняючи використану ними пам'ять.

#### Значущі і посилальні типи

Тип даних визначає обсяг пам'яті, необхідний для їх зберігання. Тип також визначає, який вид пам'яті використовується для розміщення даних – стек чи купа. Всі типи ділять на дві великі категорії: *типи-значення* (або *значущі типи*) і *типи-посилання* (або *посилальні типи*). Екземпляри цих типів розміщуються у пам'яті по-різному.

*Значущі типи* потребують тільки однієї ділянки пам'яті, у якій зберігається реальне значення. Вони можуть розміщуватися і в стеку, і в купі. Змінні цього типу безпосередньо представляють значення, яке зберігається у відповідній ділянці пам'яті. Прикладами типів-значень є всі числові типи (int, double тощо).

*Посилальні типи* потребують двох ділянок пам'яті: перша містить реальні дані (зазвичай, вона більша) і завжди розташована у купі, а друга є посиланням, яке вказує на ці дані в купі. Посилання може розміщуватися як у стеку, так і в купі. Змінна цього типу представляє не дані, а посилання на них. У той час, як дані можуть займати значний обсяг пам'яті, посилання завжди мале: 32 чи 64 біти, залежно від розрядності системи. Використання цієї моделі розміщення даних забезпечує ефективнішу роботу, так як при обміні даними між різними методами не потрібно копіювати великі обсяги даних об'єкта – достатньо передавати короткі посилання.

Таблиця 2.7 групує типи даних на дві категорії – значущі та посилальні типи, окремо для попередньо визначених та користувацьких типів.

Таблиця №2.7. Категорії типів даних у C#

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Значущі типи** |  |  | **Посилальні типи** |
| **Попередньо визначені типи** | sbyte byte short ushort long ulong decimal | float int  char | double uint bool | object string  dynamic |
| **Типи, визначені користувачем** | struct enum |  |  | array class interface delegate |

1. Під "сміттям" мають на увазі вже не потрібні об'єкти у пам'яті. [↑](#footnote-ref-1)