**BufferedInputStream**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 8](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=8), Лекция 6

— Привет, Амиго! Сегодня я расскажу тебе немного интересных вещей про класс BufferedInputStream, но начнем мы с «***обертки***» и «***мешка сахара***».

— Это что еще за «обертка» и «мешок сахара»?

— Это метафоры. Слушай. Итак…

Паттерн проектирования «Обёртка» (Wrapper или Decorator) – это довольно простой и удобный механизм расширения функциональности объектов.

[](https://cdn.javarush.ru/images/article/5c820456-3179-4601-9b51-274034605a14/original.jpeg)

Пусть у нас есть класс Cat с двумя методами getName и setName:

|  |  |
| --- | --- |
| **Код на Java** | **Описание** |
| class Cat  {  private String name;  public Cat(String name)  {  this.name = name;  }  public String getName()  {  return this.name;  }  public void setName(String name)  {  this.name = name;  }  } | Класс Кот(Cat) имеет два метода: getName & setName |
| public static void main(String[] args)  {  Cat cat = new Cat("Васька");  printName(cat);  }  public static void printName(Сat cat)  {  System.out.println(cat.getName());  } | Пример использования.  В консоль будет выведена строка «Васька». |

Допустим нам нужно перехватить вызов методов у объекта **cat** и, возможно, внести туда небольшие изменения. Для этого нам понадобится ***обернуть*** его в свой класс-обертку.

**Если мы хотим «обернуть» вызовы методов какого-то объекта своим кодом, то нам нужно:**

**1)** Создать свой класс-обертку и унаследоваться от класса/интерфейса, для которого делаем обертку.

**2)** Передать оборачиваемый объект в конструктор нашего класса.

**3)** Переопределить все методы в нашем новом классе, и вызвать в них методы оборачиваемого объекта.

**4)** Внести свои изменения «по вкусу»: менять результаты вызовов, параметры или делать что-то еще.

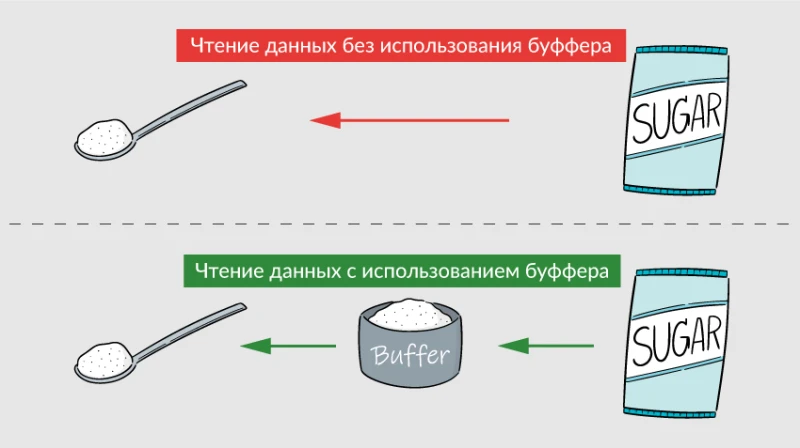
В примере ниже мы перехватываем вызов метода getName у объекта cat и немного меняем его результат.

|  |  |
| --- | --- |
| **Код на Java** | **Описание** |
| class Cat  {  private String name;  public Cat(String name)  {  this.name = name;  }  public String getName()  {  return this.name;  }  public void setName(String name)  {  this.name = name;  }  } | Класс Кот(Cat) содержит два метода – получить имя и установить имя. |
| class CatWrapper extends Cat  {  private Cat original;  public CatWrapper (Cat cat)  {  super(cat.getName());  this.original = cat;  }  public String getName()  {  return "Кот по имени " + original.getName();  }  public void setName(String name)  {  original.setName(name);  }  } | Класс-обертка. Класс не хранит никаких данных, кроме ссылки на оригинальный объект. Класс в состоянии «пробрасывать» вызовы оригинальному объекту (setName), переданному ему в конструкторе.А также «перехватывать» эти вызовы и модифицировать их параметры и результаты. |
| public static void main(String[] args)  {  Cat cat = new Cat("Васька ");  Cat catWrap = new CatWrapper (cat);  printName(catWrap);  }  public static void printName(Cat named)  {  System.out.println(named.getName());  } | Пример использования.  В консоль будет выведена строка «Кот по имени Васька». |

Т.е. мы тихонечко подменяем каждый оригинальный объект на объект-обертку, в который уже передаем ссылку на оригинальный объект. Все вызовы методов у обертки идут к оригинальному объекту, и все работает как часы.

— Мне понравилось. Решение несложное и функциональное.

— Еще я расскажу тебе про «мешок сахара», но это не паттерн, а метафора. Метафора к слову буфер и буферизация. Что же такое буферизация и зачем она нужна?

[](https://cdn.javarush.ru/images/article/0701c44b-f64d-4b22-9ae0-4c4298b2403d/original.jpeg)

Допустим, сегодня очередь Риши готовить, а ты ему помогаешь. Риши еще нет, а я хочу выпить чай и прошу тебя принести мне ложечку сахара. Ты пошел в подвал, там стоит мешок с сахаром. Ты можешь принести мне целый мешок, но мешок мне не нужен. Мне нужна только одна ложка. Тогда ты, как хороший робот, набрал одну ложку и принес мне. Я добавила ее в чай, но все равно не очень сладко. И я попросила у тебя еще одну. Ты опять сходил в подвал и принес еще ложку. Потом пришла Элли, и я попросила тебя принести сахара для нее… Это все слишком долго и неэффективно.

Пришел Риша, посмотрел на все это и попросил тебя принести ему полную сахарницу сахара. Потом я и Элли стали просить сахар у Риши. Он просто давал его нам из сахарницы, и все.

То, что произошло после появления Риши называется **буферизацией**, а сахарница – это буфер. Благодаря буферизации «клиенты» могут читать данные из буфера маленькими порциями, а буфер, чтобы сэкономить время и силы, читает их из источника большими порциями.

— Классный пример, Ким. Я все понял. Просьба ложки сахара – это аналог чтения из потока одного байта.

— Да. Класс **BufferedInputStream** – классический представитель обертки-буфера. Он – класс-обертка над InputStream. При чтении данных из него, он читает их из оригинального InputStream’а большими порциями в буфер, а потом отдает из буфера потихоньку.

— Отлично. Все понятно. А буферы для записи бывают?

— Да, конечно.

— А можно пример?

— Представь себе мусорное ведро. Вместо того, чтобы каждый раз ходить выбрасывать мусор на улице в дезинтегратор, ты просто выкидываешь его в мусорное ведро. А Скрафи раз в две недели выносит его на улицу. Классический буфер.

[](https://cdn.javarush.ru/images/article/1597887d-a456-447c-be41-409320cfb68c/original.jpeg)

— Как интересно. И гораздо понятнее, кстати, чем с мешком сахара.

— А метод flush() – это вынести мусор немедленно. Можно использовать перед приходом гостей.