❓なぜ `Print p = new PrintBanner("Hello");` にするのか？

これは **インターフェース（もしくは抽象クラス）である `Print` 型を使うことで、柔軟性と拡張性を得るため**です。

🎯 意図：上位型で変数を定義する目的

```java
Print p = new PrintBanner("Hello");
```

このように**「上位型（インターフェースや抽象クラス）で宣言」し、「下位クラスでインスタンス化」**することで、次のようなメリットがあります。

✅ 利点と意図の解説

1. ポリモーフィズムを活用できる

- `Print` は `printWeak()` や `printStrong()` という「操作の契約（インターフェース）」を定義しています。
- クライアントコード（`Main` クラス）は `PrintBanner` **の詳細な実装に依存しない。**
- 将来的に `Print` を実装した別のクラス（例：`FancyPrint`）に差し替えてもコードが壊れない。

```java
Print p = new FancyPrint("Hello"); // Print を実装していれば差し替え可能
```

---

2. コードの再利用性と保守性が向上する

- 関数に引数として `Print` 型を使えば、どんな `Print` 実装でも受け取れる：

```java
void show(Print p) {
    p.printStrong();
}
```

- こうすることで**「共通インターフェースに基づいた設計」**ができ、**柔軟な拡張が可能**になります。

---

3. 依存の方向性を制御できる（依存性逆転の原則）

- 高レベルモジュール（Mainなど）は、低レベルの具体実装（PrintBanner）に直接依存しない。
- よって、クラス間の結合度が下がり、単体テストや変更に強くなります。

🚫 なぜ `PrintBanner p = new PrintBanner("Hello");` にしないのか？

この書き方でも動作はしますが、**具体クラスに依存してしまい**、以下のようなデメリットがあります：

- 他の `Print` 実装に差し替えが効かない。
- `Main` クラスが `PrintBanner` に強く依存し、変更に弱くなる。
- `PrintBanner` 固有のメソッドに依存しやすくなり、抽象化の意図が薄れる。

💡 例えるなら：

- `Print` は「リモコンのインターフェース」。
- `PrintBanner` は「実際のテレビ」。
- リモコン（`Print`）さえあれば、テレビの詳細が `Sony` だろうと `Panasonic` だろうと関係なく操作できる。

🧠 補足：型と振る舞い

あなたの理解も非常に正しいです：

> 「これは上位のクラスのタイプでインスタンス化することで下位クラスの振る舞いをする？」

まさにこれが **ポリモーフィズム**です。宣言は `Print` 型でも、実体が `PrintBanner` なので `PrintBanner` の実装が呼び出されます。

✅ 結論

`Print p = new PrintBanner("Hello");` とする理由は：

- **抽象化を維持しながら具象に依存しない設計**を実現するため
- **ポリモーフィズム**を活用して、柔軟性と拡張性を高めるため
- **関心の分離**と**変更に強いコード**を作るため