**✅ Quick Comparison**

| **Feature** | **abstract class** | **interface** |
| --- | --- | --- |
| Keyword | abstract class | interface |
| Inheritance type | Single inheritance only | Multiple inheritance via multiple interfaces |
| Method types allowed | Abstract & concrete (with body) | Abstract (Java 7) + default/static (Java 8+) |
| Constructors | ✅ Yes | ❌ No |
| Fields/Variables | ✅ Can have fields (incl. non-final) | 🔒 Only public static final constants |
| Access modifiers | ✅ Can be private, protected, etc. | ❌ All methods are public implicitly |
| When to use | When you want a **base class with default behavior** | When you want to **define capabilities or contracts** |

**🧠 Think of it like:**

* 🧱 **abstract class** = a **foundation**. You can give partial implementation and reuse logic.
* 🎯 **interface** = a **contract**. Tells what a class *can do*, not *how* it does it.

**🔨 Example: abstract class**

java

CopyEdit

public abstract class Bidder {

private String name;

public Bidder(String name) {

this.name = name;

}

public abstract void placeBid(BigDecimal amount);

public void showName() {

System.out.println("Bidder: " + name);

}

}

✅ Can have a constructor, fields, concrete + abstract methods.

**🧩 Example: interface**

java

CopyEdit

public interface AutoBiddable {

void triggerAutoBid();

}

✅ Only method signatures, unless using default or static methods (Java 8+)

**⚙️ Real-world usage in your auction system:**

* Use an **abstract class** like AbstractAuctionService to hold shared logic for different types of auctions (English, Dutch, etc.).
* Use **interfaces** like AutoBiddable, Trackable, or Notifiable to declare optional capabilities across classes.

**✅ Quick Tip:**

If you're just defining **what should be done**, use an **interface**.  
If you're also providing **some reusable code**, go with an **abstract class**.