## Catch()

```
const cart = ["shoes", "pants", "kurta"];

createOrder(cart)
  .then(function (orderId) {
    // ☑ Step 1: Order created successfully
    console.log("🛒 Order created with ID:", orderId);
    return orderId;
  })
  .catch(function (err) {
    // ⚠ This catch handles any error in the above step ONLY
    console.log("❌ Error in createOrder:", err.message);

    // ⚠ Returning `null` to allow chain to continue gracefully
    // Instead of stopping the chain completely, we return a fallback
    return null;
  })
  .then(function (orderId) {
    // 🔁 Step 2: Continue chain even if order creation failed
    if (orderId === null) {
      console.log("⚠ Skipping payment as order creation failed.");
      return;
    }

    // ☑ If orderId is valid, proceed to payment
    return proceedToPayment(orderId);
  })
  .then(function (paymentInfo) {
    // 💳 Step 3: Handle payment success
    if (paymentInfo) {
      console.log("☑", paymentInfo);
    }
  })
  .catch(function (err) {
    // ⚠ Final safety net: handles any error in payment or below
    console.log("❌ Final error handler:", err.message);
  });


// 🌟 Producer function - simulates placing an order
function createOrder(cart) {
  return new Promise(function (resolve, reject) {
    // 🧪 Step 1: Validate cart
    if (!validateCart(cart)) {
      const err = new Error("Cart is not Valid");
      reject(err); // ❌ Reject promise if cart invalid
    }

    // 🛒 Step 2: Order placed (simulated)
    const orderId = "12345";
    resolve(orderId); // ☑ Resolve with mock orderId
```

```
  });
}

// 💳 Simulate payment processing
function proceedToPayment(orderId) {
  return new Promise(function (resolve, reject) {
    // Simulate async success
    resolve("💳 Payment Successful for Order ID: " + orderId);
  });
}

// ☑️ Mock cart validation logic
function validateCart(cart) {
  // Toggle this to false to simulate cart validation failure
  return Array.isArray(cart) && cart.length > 0;
}
```

---

### 🔍 How `.catch()` Works in Promise Chaining:

- 🧱 `.catch()` **only handles errors that occur *above* it** in the chain.
- 🎯 You can **place** `.catch()` **anywhere** to handle errors locally and **continue** the chain afterward.
- 🛡️ A **final** `.catch()` at the end acts as a **global error handler**, like a safety net.

---

### ☑️ Summary:

| Situation | Behavior |
|---|---|
| Error before first `.catch()` | That `.catch()` handles it |
| Error after `.catch()` | Will skip `.catch()` and continue unless another `.catch` |
| Return from `.catch()` | Allows the chain to **gracefully recover** |
| No `.catch()` at all | ❗ Unhandled promise rejection (runtime warning) |

---

Absolutely! Here's a **polished and well-commented version** of your promise chaining example with a detailed explanation of how `.catch()` works and emojis to make it more engaging and readable:

---

```
const cart = ["shoes", "pants", "kurta"];

createOrder(cart)
  .then(function (orderId) {
    // ☑️ Step 1: Order created successfully
    console.log("🛒 Order created with ID:", orderId);
    return orderId;
  })
  .catch(function (err) {
    // ⚠️ This catch handles any error in the above step ONLY
```

```
    console.log("❌ Error in createOrder:", err.message);

    // ⚠️ Returning `null` to allow chain to continue gracefully
    // Instead of stopping the chain completely, we return a fallback
    return null;
  })
  .then(function (orderId) {
    // 🔁 Step 2: Continue chain even if order creation failed
    if (orderId === null) {
      console.log("⚠️ Skipping payment as order creation failed.");
      return;
    }

    // ☑️ If orderId is valid, proceed to payment
    return proceedToPayment(orderId);
  })
  .then(function (paymentInfo) {
    // 💳 Step 3: Handle payment success
    if (paymentInfo) {
      console.log("☑️", paymentInfo);
    }
  })
  .catch(function (err) {
    // ⚠️ Final safety net: handles any error in payment or below
    console.log("❌ Final error handler:", err.message);
  });


// 🏭 Producer function - simulates placing an order
function createOrder(cart) {
  return new Promise(function (resolve, reject) {
    // 🧪 Step 1: Validate cart
    if (!validateCart(cart)) {
      const err = new Error("Cart is not Valid");
      reject(err); // ❌ Reject promise if cart invalid
    }

    // 🛒 Step 2: Order placed (simulated)
    const orderId = "12345";
    resolve(orderId); // ☑️ Resolve with mock orderId
  });
}

// 💸 Simulate payment processing
function proceedToPayment(orderId) {
  return new Promise(function (resolve, reject) {
    // Simulate async success
    resolve("💳 Payment Successful for Order ID: " + orderId);
  });
}

// ☑️ Mock cart validation logic
function validateCart(cart) {
  // Toggle this to false to simulate cart validation failure
```

```
  return Array.isArray(cart) && cart.length > 0;
}
```

---

### 🔍 How `.catch()` Works in Promise Chaining:

- 🧱 `.catch()` **only handles errors that occur *above* it** in the chain.
- 🎯 You can **place** `.catch()` **anywhere** to handle errors locally and **continue** the chain afterward.
- 🛡️ A **final** `.catch()` at the end acts as a **global error handler**, like a safety net.

---

### ☑️ Summary:

| Situation | Behavior |
|---|---|
| Error before first `.catch()` | That `.catch()` handles it |
| Error after `.catch()` | Will skip `.catch()` and continue unless another `.catch` |
| Return from `.catch()` | Allows the chain to **gracefully recover** |
| No `.catch()` at all | ❗ Unhandled promise rejection (runtime warning) |

---

**multiple** `.catch()` **blocks** to handle errors **at different stages** of a promise chain—giving you **granular control** over error recovery 🛠️💡:

---

### 🧪 **Scenario**:

You want to:

1. Create an order 🛒
2. Proceed to payment 💳
3. Show order summary 📦

We'll simulate failures at different points and catch them accordingly.

---

### ☑️ **Code with Multiple** `.catch()` **Blocks:**

```
const cart = ["shoes", "pants", "kurta"];

createOrder(cart)
  .then(function (orderId) {
    console.log("🛒 Order created with ID:", orderId);
    return orderId;
  })
  .catch(function (err) {
    // 🟤 Handles error in createOrder ONLY
    console.log("❌ Error in creating order:", err.message);
```

```
    // Decide whether to stop or continue
    return null; // Let's continue the chain gracefully
  })
  .then(function (orderId) {
    if (orderId === null) {
      console.log("⚠️ Skipping payment due to order failure.");
      return null;
    }

    // Proceed to payment
    return proceedToPayment(orderId);
  })
  .catch(function (err) {
    // 🟤 Handles error in proceedToPayment
    console.log("❌ Error in payment:", err.message);

    // Optional fallback or continue
    return "FailedPayment"; // Pass fallback to next then
  })
  .then(function (paymentInfo) {
    if (paymentInfo === null) {
      console.log("⚠️ No payment info available.");
      return;
    }

    console.log("✅ Payment Response:", paymentInfo);
    return showOrderSummary(); // Proceed to final step
  })
  .catch(function (err) {
    // 🟤 Handles error in showOrderSummary
    console.log("❌ Error in showing summary:", err.message);
  });


// 🎯 Function definitions

function createOrder(cart) {
  return new Promise(function (resolve, reject) {
    if (!validateCart(cart)) {
      return reject(new Error("Cart is invalid"));
    }

    const orderId = "12345"; // Simulated order ID
    resolve(orderId);
  });
}

function proceedToPayment(orderId) {
  return new Promise(function (resolve, reject) {
    // Simulate failure
    const isPaymentSuccessful = false;

    if (!isPaymentSuccessful) {
      return reject(new Error("Payment failed"));
```

```
    }

    resolve("💳 Payment Successful for Order ID: " + orderId);
  });
}

function showOrderSummary() {
  return new Promise(function (resolve, reject) {
    resolve("📦 Order Summary Displayed");
  });
}

function validateCart(cart) {
  return Array.isArray(cart) && cart.length > 0;
}
```

---

### 📌 Key Points:

- ☑ **Each `.catch()` only handles errors in the promise chain *above* it.**
- 🔄 You can **continue the chain after a `.catch()`** by returning a fallback.
- 🌓 Use **multiple `.catch()` blocks** to isolate error handling by stage.

---

### 🔁 Want to simulate an error in `showOrderSummary()` too?

Just change this:

```
resolve("📦 Order Summary Displayed");
```

to:

```
reject(new Error("Summary service unavailable"));
```

---