SLIDE:7

Rainforest Simulation — Problem Statement

● Simulating a rainforest involves millions of trees.

● Each tree stores species data (name, color, texture, growth rate) and position/height.

● Issue: Species data is repeated for every tree → enormous memory consumption and poor performance.

Rainforest Simulation — Problem Statement (সমস্যার বিবৃতি)

- রেইনফরেস্ট বা জঙ্গলের একটি সিমুলেশন করলে সেখানে লক্ষ লক্ষ গাছ **(millions of trees)** থাকতে পারে।
- প্রতিটি গাছের জন্য কিছু ডাটা রাখা হয়, যেমন:
  - **Species data (প্রজাতির তথ্য):** নাম, রঙ, টেক্সচার, গ্রোথ রেট (বৃদ্ধির হার)
  - **Position/Height (অবস্থান/উচ্চতা)**
- সমস্যাটা হচ্ছে:
  - প্রতিটি গাছের জন্য একই species data বারবার কপি করে রাখা হচ্ছে।
  - এর ফলে প্রচুর মেমরি খরচ হচ্ছে এবং পারফরম্যান্স খারাপ হচ্ছে।

🌴 Flyweight Design Pattern — সমাধান

সমস্যা **recap:**

- প্রতিটি গাছের species data (নাম, রঙ, texture, growth rate) প্রতিটি **instance-**এর জন্য আলাদাভাবে রাখা হচ্ছে।
- এতে একই ডাটা বারবার কপি হয়ে মেমরি নষ্ট হচ্ছে।

🛠️ Flyweight Pattern কী করে?

- **Flyweight** প্যাটার্ন এমন ডাটা আলাদা করে রাখে যেগুলো অনেক অবজেক্টের জন্য একই রকম থাকে (shared data → intrinsic state)।
- প্রতিটি অবজেক্টের মধ্যে শুধু **unique data** (extrinsic state) রাখা হয়।
- এতে মেমরি অনেক বাঁচে এবং পারফরম্যান্স বাড়ে।

🌳 Rainforest Simulation এ ব্যবহার

১. Intrinsic State (শেয়ার করা ডাটা):

- Species data → নাম, রঙ, texture, growth rate
  👉 এগুলো একবারই মেমরিতে রাখা হবে, সব গাছ একই species data reference ব্যবহার করবে।

২. **Extrinsic State (**গাছভেদে আলাদা ডাটা**):**

- Position, Height
  👉 প্রতিটি গাছের জন্য আলাদাভাবে রাখা হবে।

🖼️ উদাহরণ **(Conceptual)**

ধরুন ১০ লাখ গাছ আছে, আর মাত্র ১০০ species।

Flyweight ছাড়া**:**

- প্রতিটি গাছ = species data + position + height
- অর্থাৎ species data (নাম, রঙ ইত্যাদি) ১০ লাখ বার ডুপ্লিকেট।

Flyweight সহ**:**

- species data (১০০ species) = মাত্র ১০০ বার মেমরিতে রাখা হবে।
- প্রতিটি গাছ শুধু species reference + position + height রাখবে।

👉 এতে হাজার গুণ মেমরি কম লাগবে

⚡ সুবিধা

- মেমরি ব্যবহার অনেক কমে যাবে।
- একই species data একবারই তৈরি হবে → ডাটা consistency বাড়ে।
- অনেক বড় সিমুলেশনও দ্রুত চলবে।

SLIDE:8-11

Before Code (Flyweight ছাড়া)

- প্রতিটি গাছের অবজেক্ট নিজের **texture (appearance)** এবং **position (x, y)** আলাদাভাবে রাখছে।
- এখানে `texture` আসলে শেয়ার করা ডাটা (অনেক গাছের একই texture থাকে), কিন্তু তাও প্রতিটি অবজেক্টে কপি হয়ে যাচ্ছে।

- এর মানে হলো:
    - **Duplicate data** → একই texture বারবার store হচ্ছে।
    - **Memory Waste** → হাজারো গাছ থাকলে texture data লাখ লাখ বার কপি হবে।
- `draw()` মেথড গাছের texture আর coordinate প্রিন্ট করে (বা আঁকে), কিন্তু মেমরি ব্যবহারে অনেক অপচয় হয়।

সহজভাবে: প্রতিটি গাছ নিজের species data বারবার রাখে, তাই অতিরিক্ত মেমরি খরচ হয়।

Before Code (Flyweight ছাড়া)

- প্রতিটি গাছের অবজেক্ট নিজের **texture (appearance)** এবং **position (x, y)** আলাদাভাবে রাখছে।
- এখানে `texture` আসলে শেয়ার করা ডাটা (অনেক গাছের একই texture থাকে), কিন্তু তাও প্রতিটি অবজেক্টে কপি হয়ে যাচ্ছে।

- এর মানে হলো:
    - **Duplicate data** → একই texture বারবার store হচ্ছে।
    - **Memory Waste** → হাজারো গাছ থাকলে texture data লাখ লাখ বার কপি হবে।
- `draw()` মেথড গাছের texture আর coordinate প্রিন্ট করে (বা আঁকে), কিন্তু মেমরি ব্যবহারে অনেক অপচয় হয়।

সহজভাবে: প্রতিটি গাছ নিজের species data বারবার রাখে, তাই অতিরিক্ত মেমরি খরচ হয়।

SLIDE:12

Advantages: ● Memory Efficiency – shared state avoids duplication ● Performance Boost – fewer objects, faster execution ● Reduced Object Creation – reuses existing flyweights ● Centralized Management – factory controls shared objects ● State Separation – clear split of intrinsic vs extrinsic data

Flyweight Pattern-এর সুবিধা (Advantages)

1. Memory Efficiency – shared state avoids duplication

মেমরির ব্যবহার অনেক কম হয়, কারণ সাধারণ (shared) ডাটা বারবার কপি না হয়ে একবারই রাখা হয়।

- উদাহরণ: হাজার গাছের একই রঙ বা টেক্সচার থাকলে সেটা একবারই মেমরিতে থাকবে, প্রতিটি গাছ সেটা রেফারেন্স করবে।

2. Performance Boost – fewer objects, faster execution

👉 অবজেক্ট সংখ্যা কমে যায়, তাই প্রোগ্রাম দ্রুত চলে।

- প্রতিটি গাছের জন্য নতুন অবজেক্ট বানাতে হয় না → পুরানোটা শেয়ার করা হয়।
- অবজেক্ট তৈরি ও ম্যানেজ করার খরচ কমে যায় → এক্সিকিউশন স্পিড বেড়ে যায়।

3. Reduced Object Creation – reuses existing flyweights

👉 নতুন অবজেক্ট কম তৈরি হয়, আগের তৈরি হওয়া flyweight অবজেক্টগুলো বারবার ব্যবহার হয়।

- Factory চেক করে → species আগে থেকে আছে কিনা।
- থাকলে নতুন বানায় না, বরং আগেরটা রিইউজ করে।

4. Centralized Management – factory controls shared objects

👉 সব shared ডাটা এক জায়গায় **(Factory class)** ম্যানেজ করা হয়।

- এতে কোডের কনসিস্টেন্সি বাড়ে।
- species data আপডেট বা পরিবর্তন করলে সব জায়গায় একই পরিবর্তন প্রতিফলিত হয়।

5. State Separation – clear split of intrinsic vs extrinsic data

👉 Flyweight প্যাটার্নে ডাটা দুই ভাগে ভাগ করা হয়:

- **Intrinsic (shared) state** → species data (নাম, রঙ, texture, growth rate)।
- **Extrinsic (unique) state** → গাছের আলাদা আলাদা data (position, height)।

এতে প্রোগ্রামের স্ট্রাকচার পরিষ্কার হয়, কোনটা শেয়ারড আর কোনটা ইউনিক সেটা সহজে বোঝা যায়

SLIDE:13
Disadvantages
- Increased Complexity – extra factory and design effort
- Client Overhead – must supply extrinsic state every time
- Immutability Requirement – shared state cannot be changed
- Debugging Difficulty – shared objects harder to trace
- Overkill for Small Systems – no real benefit with few objects

1. Increased Complexity – extra factory and design effort

👉 এই প্যাটার্ন ব্যবহার করতে গেলে কোড অনেকটা জটিল হয়ে যায়।

- আলাদা করে Factory class বানাতে হয়।
- intrinsic vs extrinsic ডাটা আলাদা করে ম্যানেজ করতে হয়।
  ছোটখাটো সিস্টেমের জন্য এটা অতিরিক্ত ঝামেলা মনে হতে পারে।

2. Client Overhead – must supply extrinsic state every time

👉 Client (ব্যবহারকারী কোড) কে প্রতিবার **unique data (extrinsic state)** দিতে হয়।

- যেমন, প্রতিটি গাছের জন্য position (x, y) আর height আলাদাভাবে দিতে হবে।
- এতে Client code এর ওপর বাড়তি চাপ পড়ে।

3. Immutability Requirement – shared state cannot be changed

👉 Flyweight-এ shared ডাটা (intrinsic state) **immutable** রাখতে হয়।

- কারণ, একবার যদি কেউ species data পরিবর্তন করে, তাহলে যেসব গাছ সেটি ব্যবহার করছে সবার ওপর প্রভাব পড়বে।
- তাই shared state পরিবর্তন করার সুযোগ থাকে না।

4. Debugging Difficulty – shared objects harder to trace

👉 Debug করা কঠিন হয়।

- একই flyweight অবজেক্ট অনেক গাছ ব্যবহার করছে → কোন গাছ কোন ডাটা ব্যবহার করছে সেটা ট্র্যাক করা কঠিন হয়ে যায়।
- প্রোগ্রামারদের জন্য সমস্যা তৈরি করতে পারে।

5. Overkill for Small Systems – no real benefit with few objects

👉 যদি সিস্টেমে অবজেক্ট সংখ্যা কম হয়, তাহলে Flyweight ব্যবহার করে তেমন কোনো সুবিধা পাওয়া যায় না।

- বরং design জটিল হয়ে যায়।
- তাই এই প্যাটার্ন মূলত বড় সিস্টেম **(**যেখানে হাজার**/**লক্ষ অবজেক্ট আছে**)** এর জন্য উপযুক্ত।

⚡সারকথা: Flyweight প্যাটার্ন বড় সিস্টেমে অনেক সুবিধা দেয়, কিন্তু ছোট সিস্টেমে উল্টে জটিলতা বাড়ায়**, client code** কঠিন করে তোলে এবং **debug** করা কষ্টকর হয়**।**

SLIDE:14

1. Text Editors (MS Word, Notepad++)

👉 টেক্সট এডিটরে প্রতিটি অক্ষর (A–Z), ফন্ট, আর স্টাইল বারবার লিখতে হয়।

- এখানে অক্ষর**,** ফন্ট**,** স্টাইল → shared flyweight (intrinsic state)**।**
- প্রতিটি অক্ষরের পজিশন**,** সাইজ**,** ফরম্যাটিং **(bold, italic** ইত্যাদি**)** → extrinsic state, যেটা আলাদাভাবে রাখা হয়।
  ➡️ এর ফলে মেমরি অনেক বাঁচে এবং লক্ষ-কোটি অক্ষরও সহজে ম্যানেজ করা যায়।

2. Game Development (Forests, Cities, Chess)

👉 গেমে হাজার হাজার অবজেক্ট থাকে, যেমন:

- **Forests:** অনেক গাছ (Tree) → species data (রঙ, texture) শেয়ার হয়।
- **Cities:** ঘরবাড়ি (House), টাইলস (Tiles) → একই মডেল অনেক জায়গায় শেয়ার হয়।
- **Chess Game:** একই ধরনের ঘুটি (Pawn, King, Queen) → মডেল শেয়ার হয়, কিন্তু পজিশন (extrinsic state) আলাদা থাকে।
  ➡️ এতে গেম স্মুথ চলে, কারণ মডেল ডুপ্লিকেট না করে শেয়ার করা হয়।

3. GUI Systems (Icons, Buttons, Widgets)

👉 GUI সিস্টেমে একই আইকন বা বাটন অনেক জায়গায় ব্যবহার করা হয়।

- **Icon/Image** → shared flyweight (intrinsic)।
- প্রতিটি আইকন/বাটনের পজিশন**,** সাইজ**,** স্টেট **(enabled/disabled)** → extrinsic।
  ➡️ এতে একই ইমেজ/আইকন একবার লোড করলেই UI-এর যেকোনো জায়গায় রিইউজ করা যায়।

⚡সারকথা:

- **Text Editor** এ → অক্ষর, ফন্ট শেয়ার হয়।
- **Game Development** এ → গাছ, ঘরবাড়ি, ঘুটি ইত্যাদি শেয়ার হয়।
- **GUI System** এ → আইকন, বাটন, উইজেট শেয়ার হয়।