پیشرفته — ساختار فایل‌ها و پکیج TypeScript کدها

باشد (حداقل یک **ماژول** هر فایل، Duplicate identifier نکتهٔ مهم: برای جلوگیری از خطای `export` یا `import`).

می‌توانید یک `tsconfig.json` ساده هم داشته باشید:

```
{
 "compilerOptions": {
  "target": "ES2020",
  "module": "ES2020",
  "moduleResolution": "Node",
  "strict": true,
  "esModuleInterop": true,
  "skipLibCheck": true,
  "forceConsistentCasingInFileNames": true
 },
 "include": ["./**/*.ts"]
}
```

01-Generics.ts

```
// 01-Generics.ts
// مثال‌های حرفه‌ای جنریک‌ها + قیود و پارامتر پیش‌فرض

// تابع جنریک ساده
export function identity<T>(value: T): T { return value; }

// محدود کردن T (Constraint)
export function prop<T extends object, K extends keyof T>(obj: T, key: K): T[K] {
 return obj[key];
}

// پارامتر جنریک پیش‌فرض
export type Box<T = string> = { value: T };

// کلاس صف جنریک
export class Queue<T> {
 private items: T[] = [];
 enqueue(item: T) { this.items.push(item); }
 dequeue(): T | undefined { return this.items.shift(); }
 peek(): T | undefined { return this.items[0]; }
}
```

```ts
// ترکیب چند جنریک
export function mapArray<T, U>(arr: T[], fn: (x: T, i: number) => U): U[] {
 const out: U[] = [];
 for (let i = 0; i < arr.length; i++) out.push(fn(arr[i], i));
 return out;
}

// Repository الگوی ساده
export interface Entity { id: string }
export class MemoryRepository<T extends Entity> {
 private data = new Map<string, T>();
 upsert(entity: T) { this.data.set(entity.id, entity); }
 findById(id: string): T | undefined { return this.data.get(id); }
}
```

02-UtilityTypes.ts

```ts
// 02-UtilityTypes.ts
// Partial, Required, Readonly, Pick, Omit, Record, Exclude, Extract,
NonNullable, ReturnType, InstanceType, Parameters, ConstructorParameters,
Awaited

export interface Todo { title: string; description: string; completed: boolean |
null }

export const t1: Partial<Todo> = { title: "Buy milk" };
export const t2: Required<Todo> = { title: "X", description: "Y", completed:
false };
export const t3: Readonly<Todo> = { title: "A", description: "B", completed:
null };
export type TodoPreview = Pick<Todo, "title" | "completed">;
export type TodoWithoutDesc = Omit<Todo, "description">;
export type FeatureFlags = Record<"dark" | "beta" | "newUI", boolean>;

export type OnlyString = Extract<string | number | null, string>; // string
export type NotNullish = NonNullable<string | undefined | null>;  // string

export function f(a: number, b: string) { return { a, b }; }
export type FReturn = ReturnType<typeof f>; // { a: number; b: string }
export type FParams = Parameters<typeof f>; // [number, string]

export class Person { constructor(public name: string) {} }
export type PInstance = InstanceType<typeof Person>;

// Awaited نمونه
```

```ts
export type Resolved<T> = T extends Promise<infer R> ? R : T;
export type R1 = Resolved<Promise<number>>; // number
```

03-Keyof-Lookup.ts

```ts
// 03-Keyof-Lookup.ts
// keyof و Lookup Types با توابع امن get/set

export interface User { id: string; name: string; age: number; role: "admin" |
"user" }

export type UserKeys = keyof User;       // "id" | "name" | "age" | "role"
export type RoleType = User["role"];     // "admin" | "user"

export function getProperty<T, K extends keyof T>(obj: T, key: K): T[K] {
 return obj[key];
}

export function setProperty<T, K extends keyof T>(obj: T, key: K, value: T[K]) {
 obj[key] = value;
}

const u: User = { id: "1", name: "Ali", age: 30, role: "admin" };
const role = getProperty(u, "role"); // تایپ دقیق
setProperty(u, "age", 31);
```

04-MappedTypes.ts

```ts
// 04-MappedTypes.ts
// بازسازی انواع، تغییر modifiers، Remap کلیدها با Template Literals

export interface Features { darkMode: () => void; newUI: () => void; version:
string }

// همهٔ کلیدها ساده: boolean
export type OptionsFlags<T> = { [K in keyof T]: boolean };
export const flags: OptionsFlags<Features> = { darkMode: true, newUI: false,
version: true };

// افزودن readonly و اختیاری کردن (Modifiers)
export type ReadonlyOptional<T> = { readonly [K in keyof T]?: T[K] };

// Conditional + Mapped: فقط توابع → boolean، بقیه بدون تغییر
export type FeatureOptions<T> = { [K in keyof T]: T[K] extends Function ? boolean :
T[K] };
```

```ts
export const advanced: FeatureOptions<Features> = { darkMode: true, newUI: false,
version: "1.0" };

// Remap با کلیدها as + Template Literals: ساخت Getterها
export type Getters<T> = {
 [K in keyof T as `get${Capitalize<string & K>}`]: () => T[K]
};

export type FeatureGetters = Getters<Features>;
// نتیجه: { getDarkMode: () => () => void; getNewUI: () => () => void;
getVersion: () => string }
```

05-ConditionalTypes.ts

```ts
// 05-ConditionalTypes.ts
// انواع شرطی + توزیعی بودن روی Union ها

export type IsString<T> = T extends string ? true : false;
export type A = IsString<string>; // true
export type B = IsString<number>; // false

// توزیعی بودن روی Union
export type ExtractBy<T, U> = T extends U ? T : never;
export type OnlyNumbers = ExtractBy<string | number | boolean, number>; // number

// Flatten و Awaited ساده
export type Flatten<T> = T extends (infer U)[] ? U : T;
export type AwaitedLike<T> = T extends Promise<infer U> ? AwaitedLike<U> : T;

// تبدیل Union به Intersection (ترفند پیشرفته)
export type UnionToIntersection<U> = (U extends any ? (x: U) => any : never)
extends (x: infer I) => any ? I : never;
export type UI = UnionToIntersection<{ a: string } | { b: number }>; // { a:
string } & { b: number }
```

06-DiscriminatedUnions.ts

```ts
// 06-DiscriminatedUnions.ts
// یونین متمایز + بررسی کامل بودن با never

export type Success = { status: "success"; data: string };
export type Failure = { status: "error"; message: string };
export type Loading = { status: "loading" };
export type APIResponse = Success | Failure | Loading;
```

```ts
export function handle(res: APIResponse) {
 switch (res.status) {
  case "success":
   return res.data.toUpperCase();
  case "error":
   return `ERR: ${res.message}`;
  case "loading":
   return "Loading...";
  default: {
   const _exhaustive: never = res; // اگر نوع جدیدی اضافه شود، اینجا خطا می‌دهد
   return _exhaustive;
  }
 }
}
```

07-NeverType.ts

```ts
// 07-NeverType.ts
// کاربردهای never: و عدم بازگشت، بن‌بست تایپی، exhaustiveness checking

// عدم بازگشت
export function fail(message: string): never { throw new Error(message); }
export function loopForever(): never { while (true) {} }

// مثال Intersection ناسازگار که به never منتهی می‌شود
export type A = { role: "admin" };
export type B = { role: "user" };
export type AB = A & B; // role: never

// برای بررسی پوشش کامل حالت‌ها استفاده از never
export type Shape = { kind: "circle"; r: number } | { kind: "square"; s: number };
export function area(shape: Shape): number {
 if (shape.kind === "circle") return Math.PI * shape.r ** 2;
 if (shape.kind === "square") return shape.s ** 2;
 const _x: never = shape; // اگر نوع جدید اضافه شود، این خط خطا می‌دهد
 return _x;
}
```

08-UnionIntersection.ts

```ts
// 08-UnionIntersection.ts
// مثال‌های بی‌خطا برای Union و Intersection

export type Admin = { role: "admin"; permissions: string[] };
export type Member = { role: "member"; points: number };
```

```ts
// Union: یکی از دو نوع
export type Person = Admin | Member;
export const p1: Person = { role: "admin", permissions: ["read"] };
export const p2: Person = { role: "member", points: 10 };

// Intersection: فقط وقتی فیلدهای مشترک سازگارند
export type WithId = { id: string };
export type WithTimestamps = { createdAt: Date; updatedAt: Date };
export type Entity = WithId & WithTimestamps;
export const e: Entity = { id: "1", createdAt: new Date(), updatedAt: new Date() };

// اگر بخواهیم Admin و Member فیلد مشترک role را ترکیب کنیم، باید سازگار شود:
export type AdminCompat = { role: string; permissions: string[] };
export type MemberCompat = { role: string; points: number };
export type SuperUser = AdminCompat & MemberCompat; // اکنون role: string
export const su: SuperUser = { role: "admin", permissions: ["*"], points: 100 };
```

09-TypeGuards.ts

```ts
// 09-TypeGuards.ts
// typeof, instanceof, in, و توابع assert نگهبانِ نوع سفارشی،

export function printId(id: string | number) {
 if (typeof id === "string") console.log(id.toUpperCase());
 else console.log(id.toFixed(2));
}

export class Dog { bark() { console.log("woof"); } }
export class Cat { meow() { console.log("meow"); } }

export function speak(a: Dog | Cat) {
 if (a instanceof Dog) a.bark(); else a.meow();
}

// in-guard
export type Fish = { swim: () => void };
export type Bird = { fly: () => void };
export function move(pet: Fish | Bird) { "swim" in pet ? pet.swim() : pet.fly(); }

// نگهبان سفارشی (Type Predicate)
export function isDog(x: unknown): x is Dog { return x instanceof Dog; }

// برای اطمینان از یک پیش‌شرط assert تابع
export function assertHas<K extends PropertyKey>(obj: unknown, key: K): asserts
obj is Record<K, unknown> {
 if (typeof obj !== "object" || obj === null || !(key in obj)) throw new
```

```
Error("Missing key");
}
```

10-InferKeyword.ts

```ts
// 10-InferKeyword.ts
// infer برای گرفتن نوع‌های میانی از توابع/آرایه‌ها/تاپل‌ها

// ReturnType سفارشی
export type MyReturnType<F> = F extends (...args: any[]) => infer R ? R : never;

export function toPair(n: number) { return [n, String(n)] as const; }
export type Pair = MyReturnType<typeof toPair>; // readonly [number, string]

// Parameters سفارشی
export type MyParameters<F> = F extends (...args: infer P) => any ? P : never;
export type ToPairParams = MyParameters<typeof toPair>; // [number]

// استخراج نوع اعضای آرایه
export type ArrayElement<A> = A extends readonly (infer T)[] ? T : never;
export type AE = ArrayElement<readonly [1, 2, 3]>; // 1 | 2 | 3

// Head/Tail تاپل
export type Head<T extends any[]> = T extends [infer H, ...any[]] ? H : never;
export type Tail<T extends any[]> = T extends [any, ...infer R] ? R : never;

// Curry (امضای ساده‌شده)
export function curry<A, B, R>(fn: (a: A, b: B) => R) {
 return (a: A) => (b: B) => fn(a, b);
}
```

11-TemplateLiteralTypes.ts

```ts
// 11-TemplateLiteralTypes.ts
// ساخت رشته‌های نوع‌دار + Remap کلیدها

export type Size = "sm" | "md" | "lg";
export type Color = "red" | "blue" | "green";
export type ButtonVariant = `${Size}-${Color}`;

export const v1: ButtonVariant = "sm-red";

// ساخت نام ایونت‌ها
export type EventName<K extends string> = `on${Capitalize<K>}`;
export type DOMEvents = EventName<"click" | "focus" | "blur">; // "onClick" |
```

```
"onFocus" | "onBlur"

// Remap کلیدهای یک نوع به Getterها
export type Getters<T> = { [K in keyof T as `get${Capitalize<string & K>}`]: () =>
T[K] };
```

12-AdvancedFunctions.ts

```
// 12-AdvancedFunctions.ts
// Overloadها، this-parameter، در امضاهای تابع و Interface

// Overload
export function combine(a: number, b: number): number;
export function combine(a: string, b: string): string;
export function combine(a: any, b: any) { return a + b; }

// this-parameter (فقط تایپی)
export function inc(this: { value: number }, step = 1) { this.value += step; }
const counter = { value: 0, inc };
counter.inc(2);

// تعریف امضای تابع در Interface
export interface Predicate<T> { (x: T): boolean }
export const isNonEmpty: Predicate<string> = (s) => s.length > 0;
```

13-index.ts

```
// 13-index.ts
// ورودی ساده که چند مثال را اجرا می‌کند (برای تست تایپ‌ها کافیست کامپایل شود)

import { identity, Queue } from "./01-Generics";
import { flags } from "./04-MappedTypes";
import { p1, p2, su } from "./08-UnionIntersection";

console.log(identity<number>(123));
const q = new Queue<string>();
q.enqueue("A");
q.enqueue("B");
console.log(q.peek());

console.log(flags.darkMode, p1.role, p2.role, su.role);
```