Programació

UT4.5. API de Java. Creació i maneig d'objectes.

Introducció

- Quan treballem amb la classe `String`, hem vist que tots el mètodes que conté es poden consultar a través de seua API.
- Ara coneixerem què és l'API de Java i a moure'ns per la documentació.
- A més vorem algunes classes definides que poden ser útils per al desenvolupament de programes futurs.

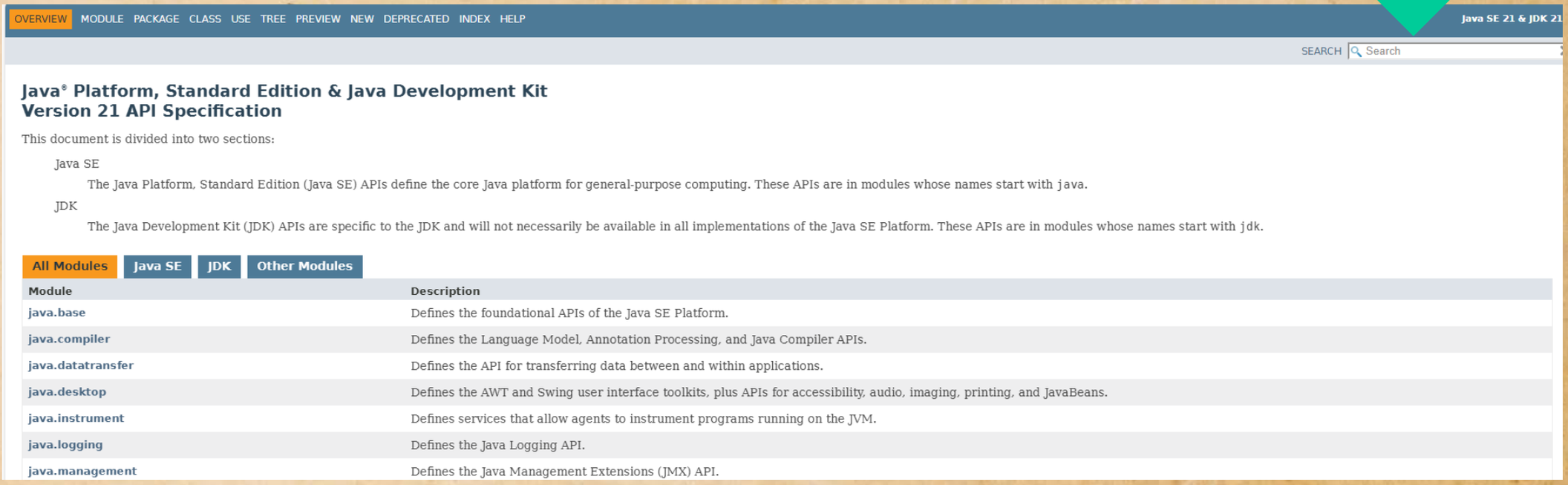
API de Java

- Java disposa d'un repositori de classes ja creades, que seran auxiliars per a la creació de programes propis.
- A este repositori l'anomenem API (Application Programming interface)
- En funció de la versió de Java consultada, este repositori varia.
- Els paquets més comuns es mantenen pràcticament en totes les versions fins a l'última versió.

API de Java (Versió 21)

https://docs.oracle.com/en/java/javase/21/docs/api/

BUSCAR

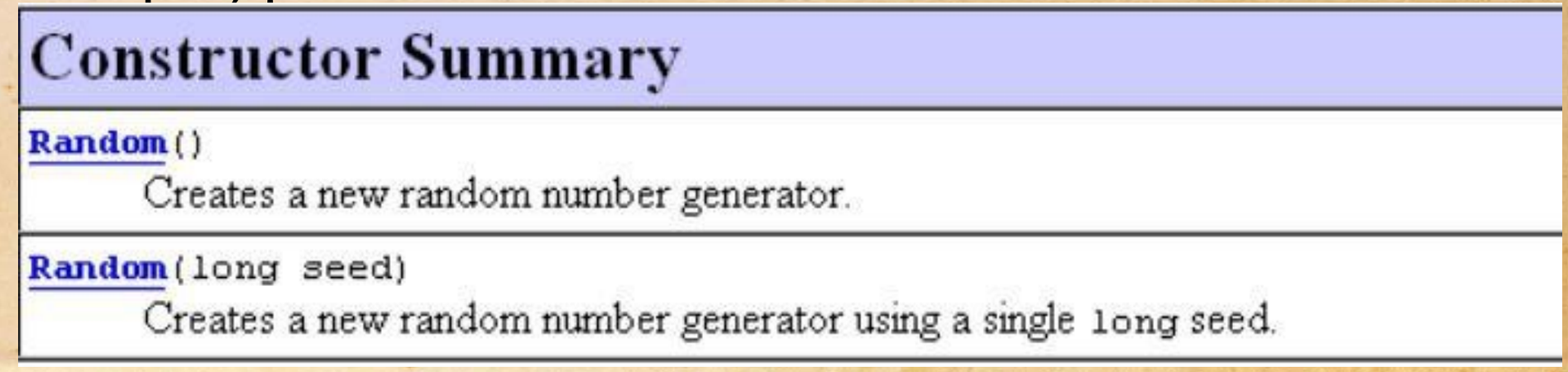


API Java. Mòduls destacats

- Dins de **`java.base`** destaquen els paquets:
  - `java.lang` - operacions essencials dels tipus de dades del llenguatge
  - `java.util` - propòsit general
  - `java.io` - entrada/eixida
- Dins de **`java.desktop`** destaca:
  - `javax.swing` - creació d'interfícies gràfiques bàsiques (A dia de hui pràcticament en desús professional, però útil per adquirir coneixements bàsics de programació d'interfícies gràfiques)

API Java

- L'apartat "Constructor Summary" mostra totes les possibilitats amb les quals una classe pot ser inicialitzada.
- Estos mètodes es coneixen com a **constructors**
- Per exemple, per a la classe `Random`:



Constructor Summary

Random()
Creates a new random number generator.

Random(long seed)
Creates a new random number generator using a single long seed.

Invocació de constructors

- Per a invocar un constructor, hem de fixar-nos en primer lloc, quants paràmetres d'entrada requereix.
- Una volta coneguda esta informació, podrem crear un objecte d'eixa classe fent ús de la paraula reservada **new.**
- Exemple per al **primer constructor de Random**:

```
Random objecteRandom = new Random ();
```

no oblidem que és una referència

Pràctica 1

- Crea un programa que genere 2 nombres reals aleatoris i els mostre per pantalla, fent ús de la classe `Random` (constructor sense paràmetres). Busca esta classe en l'API de Java i utilitza el mètode que consideres més adequat.

Pràctica 2

- Modifica el programa anterior per a que utilitze el constructor que requereix un paràmetre de tipus `long` (el segon).
- Quina diferència hi ha entre crear un objecte `Random` amb un o un altre constructor?

Mètodes estàtics

- Fins ara, per a poder invocar a mètodes de l' API, hem vist que es requereix crear un objecte de la classe.
- Existeixen també mètodes que no requereixen d'esta inicialització per a ser usats. Els identifiquem perquè apareixen marcats amb la paraula reservada **`static`.**
- Un exemple d'este tipus de mètodes els trobem per exemple a la classe **`Math`**

Mètodes estàtics:
Exemple per a la classe Math.

static long	**abs**(long a) Returns the absolute value of a long value.
static double	**acos**(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
static double	**asin**(double a) Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.

```
double resultat = Math.sqrt(36);
```

Pràctica 3

- Crea un programa que genere un nombre real aleatori i l'arrodonisca. Utilitza la classe **`Random`** i **`Math`** per a fer-ho.

Pràctica 4

- Crea un programa que calcule la hipotenusa d'un triangle. L'usuari haurà d'introduir els dos valors dels catets. Utilitza la classe `Math` per a calcular el quadrat i l'arrel quadrada.

La classe `Arrays`

- Pertany al paquet `java.util`
- Ofereix mètodes estàtics per a realitzar operacions sobre arrays:
  - Ordenació
  - Cerca
  - Còpia
  - Comparació
  - Transformació a text

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Arrays.html

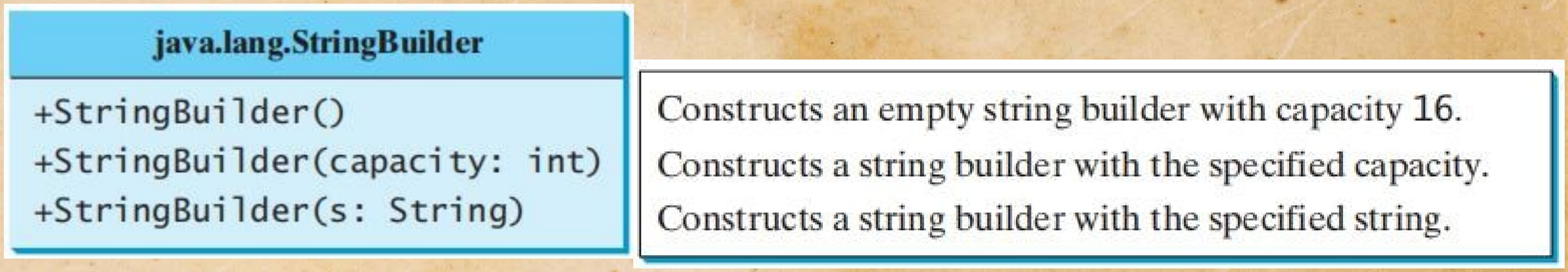
Pràctica 5

- Realitza proves sobre un array ja creat, fent ús dels diferents mètodes **`static`** que realitzen les operacions que s'han enumerat a la diapositiva anterior.

La classe `StringBuilder`

- És una classe similar a `String`, amb la diferència de que en este cas l'objecte `StringBuilder` sí que és mutable.
- Són més flexibles que `String`, ja que es pot inserir o afegir informació, a diferència d'`String`, que una volta creat l'objecte no es pot realitzar cap d'eixes accions.
- **`StringBuilder`** pot ser després transformat a **`String`** i viceversa.
- Pot ser mostrat el seu valor a través de **`System.out`**

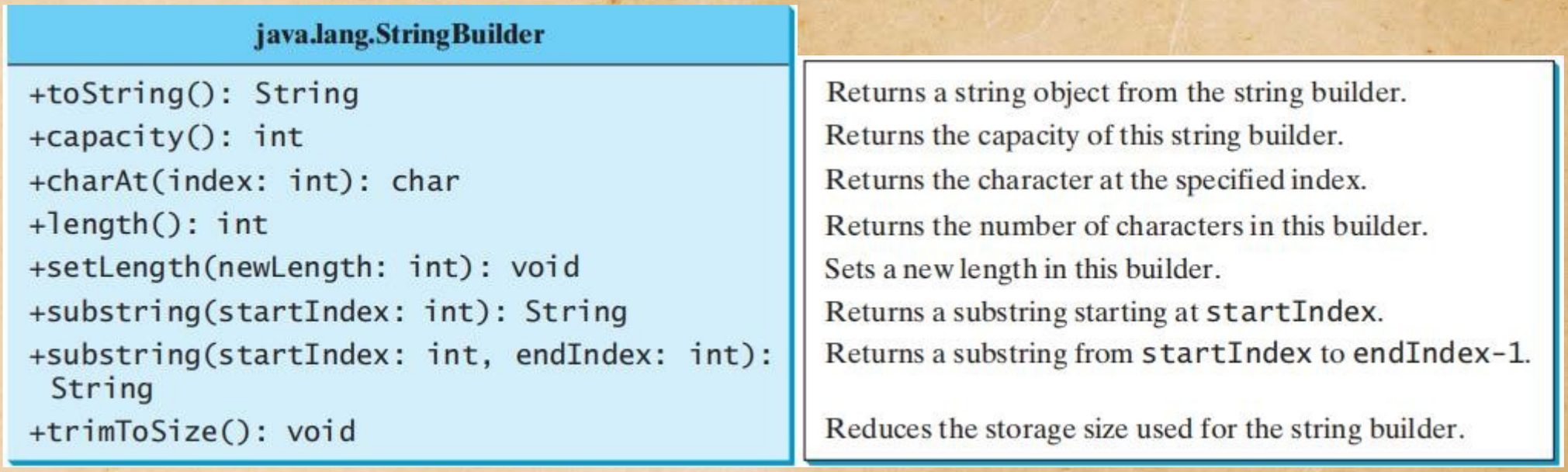
Constructors d'StringBuilder



Mètodes que modifiquen un objecte StringBuilder

java.lang.StringBuilder	
+append(data: char[]): StringBuilder	Appends a `char` array into this string builder.
+append(data: char[], offset: int, len: int): StringBuilder	Appends a subarray in `data` into this string builder.
+append(v: *aPrimitiveType*): StringBuilder	Appends a primitive type value as a string to this builder.
+append(s: String): StringBuilder	Appends a string to this string builder.
+delete(startIndex: int, endIndex: int): StringBuilder	Deletes characters from `startIndex` to `endIndex-1`.
+deleteCharAt(index: int): StringBuilder	Deletes a character at the specified index.
+insert(index: int, data: char[], offset: int, len: int): StringBuilder	Inserts a subarray of the data in the array into the builder at the specified index.
+insert(offset: int, data: char[]): StringBuilder	Inserts data into this builder at the position offset.
+insert(offset: int, b: *aPrimitiveType*): StringBuilder	Inserts a value converted to a string into this builder.
+insert(offset: int, s: String): StringBuilder	Inserts a string into this builder at the position offset.
+replace(startIndex: int, endIndex: int, s: String): StringBuilder	Replaces the characters in this builder from `startIndex` to `endIndex-1` with the specified string.
+reverse(): StringBuilder	Reverses the characters in the builder.
+setCharAt(index: int, ch: char): void	Sets a new character at the specified index in this builder.

Altres mètodes de StringBuilder



Pregunta 1

Suposant que `s1` i `s2` es declaren de la següent forma:

```
StringBuilder s1 = new StringBuilder("Java");
StringBuilder s2 = new StringBuilder("HTML");
```

Quin valor adquireix `s1` després de cada sentència?

```
a. s1.append(" is fun");
b. s1.append(s2);
c. s1.insert(2, "is fun");
d. s1.insert(1, s2);
e. s1.charAt(2);
f. s1.length();
```

```
g. s1.deleteCharAt(3);
h. s1.delete(1, 3);
i. s1.reverse();
j. s1.replace(1, 3, "Computer");
k. s1.substring(1, 3);
l. s1.substring(2);
```

Pregunta 2

Què es mostra per pantalla?

```java
public class Pregunta2 {

    public static void main(String[] args) {
        Pregunta2 pregunta2 = new Pregunta2();
        pregunta2.inicio();
    }

    public void inicio() {
        String s = "Java";
        StringBuilder builder = new StringBuilder(s);
        change(s, builder);

        System.out.println(s);
        System.out.println(builder);
    }

    private void change(String s, StringBuilder builder) {
        s = s + " and HTML";
        builder.append(" and HTML");
    }
}
```

Pregunta 3

Quin mètode de l' API d'**`StringBuilder`** usaries per obtindre un **`String`** a partir d'un **`StringBuilder`**?

Pràctica 6

- Prova en un programa nou alguns dels mètodes mostrats d'**`StringBuilder`** per a tindre encara més clar què fa cadascun.

Treballant amb dates: Importància

- **Fonamental en aplicacions reals:**
  - Gestionar esdeveniments i cites
  - Processament de transaccions financeres
  - Registre de logs i històrics d'operacions
- **Precisió i Consistència:**
  - Manipulació precisa de dates i temps
  - Evitar errors en càlculs de durades i intervals
- **Compatibilitat Internacional:**
  - Suport per zones horàries i estandardització
  - Facilitat en la conversió entre diferents formats de dates

Treballant amb dates: LocalDate, LocalTime i LocalDateTime

- **LocalDate:**
  - Representa una data (any, mes, dia)
  - Exemple: 2024-08-06
- **LocalTime:**
  - Representa una hora (hora, minut, segon)
  - Exemple: 14:30:00
- **LocalDateTime:**
  - Combina una data i una hora
  - Exemple: 2024-08-06T14:30:00

És important que investigues sobre estes classes consultant l'API de Java.

Treballant amb dates: Creació d'instàncies

- **Amb now:**
  - Obtenir la data o hora actual
  - Exemple: LocalDate.now()
- **Amb of:**
  - Crear instàncies específiques
  - Exemple: LocalDate.of(2024, 8, 6)

```
LocalDate today = LocalDate.now();
LocalDate specificDate = LocalDate.of(2024, 8, 6);
```

- **Convertir des de java.util.Date**

```
Date date = new Date();
Instant instant = date.toInstant();
LocalDate localDate = instant.atZone(ZoneId.systemDefault()).toLocalDate();
LocalDateTime localDateTime = instant.atZone(ZoneId.systemDefault()).toLocalDateTime();
```

Treballant amb dates: obtenir i modificar components individuals

- **Mètodes d'Obtenció:**
  - getYear(), getMonth(), getDayOfMonth()
  - Exemple: int year = date.getYear();
- **Mètodes de Modificació:**
  - plusDays(), minusDays() ...
  - Exemple: LocalDate nextWeek = today.plusDays(7);

```
int year = today.getYear();
LocalDate nextWeek = today.plusDays(7);
LocalDate previousMonth = today.minusMonths(1);
```

És important que investigues sobre estes classes consultant l'API de Java.

Treballant amb dates: Convertir dates a cadenes de text i viceversa

- **Format de Dates:**
  - DateTimeFormatter.ofPattern("dd/MM/yyyy")
  - Exemple:
    String formattedDate = today.format(formatter);
- **Parsing de Dates:**
  - LocalDate.parse("06/08/2024", formatter)
  - Exemple:

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
String formattedDate = today.format(formatter);
LocalDate parsedDate = LocalDate.parse("06/08/2024", formatter);
```

És important que investigues sobre estes classes consultant l'API de Java.

Treballant amb dates: Exemples

- **Calcular l'Edat a partir d'una Data de Naixement:**

```
LocalDate birthDate = LocalDate.of(2000, 1, 1);
Period age = Period.between(birthDate, LocalDate.now());
```

- **Determinar Dies entre dues Dates:**

```
LocalDate startDate = LocalDate.of(2024, 8, 1);
LocalDate endDate = LocalDate.of(2024, 8, 6);
long daysBetween = ChronoUnit.DAYS.between(startDate, endDate);
```

- **Programar un Esdeveniment Futur:**

```
LocalDate eventDate = LocalDate.now().plusMonths(2);
```

És important que investigues sobre estes classes consultant l'API de Java.

Treballant amb dates: Exercicis

- Crea una instància de LocalDate per al 25 de desembre de 2024.
- Obteniu la data actual i després afegiu-hi 10 dies.
- Si al moment actual li afegim 15 hores, quina data i hora tindrem?
- Converteix la cadena "15/08/2025" a un objecte LocalDate utilitzant el format "dd/MM/yyyy".
- A partir del LocalDate anterior, obtin un String amb el format “15-08-2025”
- Calculeu quants dies falten per al proper any nou

Classes envoltori (Wrapper classes)

- Es denominen classes envoltants o envoltori a les que permeten tractar una dada de tipus bàsic com a un objecte
- Existeixen ja que molts mètodes de l' API de Java, demanden com a paràmetre un objecte i no un tipus de dada primitiu.
- Són molt fàcil de conèixer ja que el seu nom és igual que el tipus de dada primitiu, però començant amb majúscula.

  `Boolean, Integer, Double, Float, Byte, Short, Long i Character`.

Per ampliar informació sobre els wrapper i exemples de codi

Classes envoltori: Quan s'utilitzen?

Gestió de valors nuls:

Els tipus primitius no poden ser null, però els objectes sí. Això és útil, per exemple, quan interactuem amb bases de dades (on un camp pot estar buit) o quan volem indicar que un mètode no té un resultat vàlid retornant null.

```
Integer numero = null; // Possible amb Integer, no amb int
```

Hem de gestionar adequadament estes dades per evitar excepcions com **NullPointerException**

Classes envoltori: Quan s'utilitzen?

Accés a mètodes:

Els tipus primitius no tenen mètodes associats, però les classes envoltori sí que en proporcionen molts que faciliten el treball amb valors numèrics o booleans.

```
double decimal = Double.parseDouble("3.14"); // De cadena a double

String binari = Integer.toBinaryString(10);  // "1010"
```

Classes envoltori: Quan s'utilitzen?

Ús en col·leccions i estructures de dades genèriques:

Les col·leccions en Java (com ara ArrayList, HashMap, etc.) no poden emmagatzemar tipus primitius, només objectes. Les classes envoltori permeten guardar valors numèrics o booleans en aquestes estructures.

A més, les classes envoltori són imprescindibles en programació funcional, ja que els Streams i altres operacions avançades es basen en objectes i no admeten primitius.

Classes envoltori: Quan evitar-les?

- **Rendiment crític:** Quan es fan càlculs intensius o operacions en grans volums de dades.
- **Evitar autoboxing/unboxing:** Per evitar conversions constants i errors subtils
- **Simplicitat:** Quan no necessites funcionalitats addicionals.
- **Dades que no volem que puguen valdre null:** Per evitar excepcions inesperades.

Classes envoltori: Mètodes i constants útils

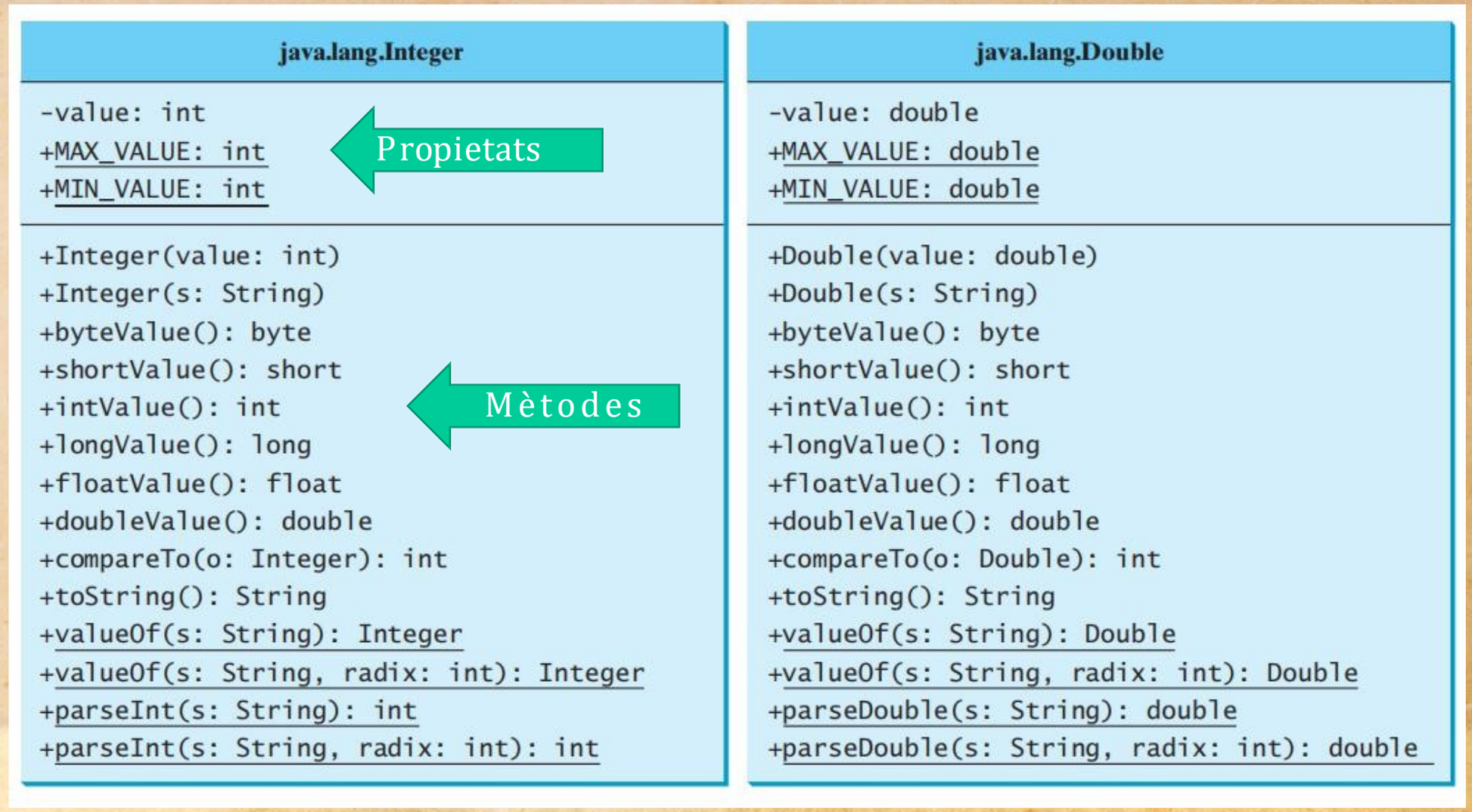
- **parseXXX() i valueOf():** Converteixen cadenes (String) a tipus numèrics (int/Integer, double/Double...)
- **compare() i compareTo():** Per a comparar objectes.
- **hashCode():** Genera un codi hash per als objectes, necessari per al seu ús en certes col·leccions.
- **toString():** Converteix l'objecte a una representació en text.

- Double.MAX_VALUE, Double.MIN_VALUE, Double.POSITIVE_INFINITY, Double.NEGATIVE_INFINITY, Double.NaN, Double.SIZE, Double.BYTES...

- **Deprecació de constructors:** Els constructors com new Integer(123) estan deprecats en versions recents de Java. S'aconsella utilitzar mètodes estàtics com valueOf().

- **Cache de valors entre -128 i 127:** Els objectes enters dins d'aquest rang són reutilitzats per a millorar el rendiment i estalviar memòria (com passava en la classe String).

Classes Integer i Double



Pregunta 4

¿Compilen les següents sentències?

```
a. Integer i = new Integer("23");
b. Integer i = new Integer(23);
c. Integer i = Integer.valueOf("23");
d. Integer i = Integer.parseInt("23", 8);
e. Double d = new Double();
f. Double d = Double.valueOf("23.45");
g. int i = (Integer.valueOf("23")).intValue();
h. double d = (Double.valueOf("23.4")).doubleValue();
i. int i = (Double.valueOf("23.4")).intValue();
j. String s = (Double.valueOf("23.4")).toString();
```

Pregunta 5

Com es converteix ....

- Un enter a String?
- Un String numèric a enter?
- Un real a un String?
- Un String numèric a real?

Pregunta 6

Què es mostra per pantalla?

```
public class Test {
  public static void main(String[] args) {
    Integer x = new Integer(3);
    System.out.println(x.intValue());
    System.out.println(x.compareTo(new Integer(4)));
  }
}
```

Pregunta 7

Què es mostra per pantalla?

```
public class Test {
  public static void main(String[] args) {
    System.out.println(Integer.parseInt("10"));
    System.out.println(Integer.parseInt("10", 10));
    System.out.println(Integer.parseInt("10", 16));
    System.out.println(Integer.parseInt("11"));
    System.out.println(Integer.parseInt("11", 10));
    System.out.println(Integer.parseInt("11", 16));
  }
}
```

Conversió automàtica entre classes envoltori i tipus primitius

- Fer la conversió d'un tipus de dada primitiu a un objecte de la classe envoltori es denomina ***boxing***
- El contrari ***unboxing***
- El compilador pot realitzar ***boxing*** i ***unboxing*** de manera automàtica (***autoboxing*** i ***autounboxing***)

Exemple de conversió automàtica

```
Integer intObject = new Integer (2);
```

EQUIVALENT A

```
Integer intObject = 2;
```

autoboxing

Pregunta 8

On es produeix *autoboxing* i on *autounboxing*? Són totes correctes?

a. `Integer x = 3 + new Integer(5);`
b. `Integer x = 3;`
c. `Double x = 3;`
d. `Double x = 3.0;`
e. `int x = new Integer(3);`
f. `int x = new Integer(3) + new Integer(4);`

Pregunta 9

Què es mostra per pantalla?

```
public class Test {
   public static void main(String[] args) {
      Double x = 3.5;
      System.out.println(x.intValue());
      System.out.println(x.compareTo(4.5));
   }
}
```

Classe Optional

La classe **Optional** és una classe envoltori en Java dissenyada per a encapsular un valor que pot o no estar present, evitant així els problemes associats amb l'ús de **null**. És molt útil en el context de POO (Programació Orientada a Objectes) per a gestionar **valors opcionalment disponibles** de manera clara i robusta, especialment en el retorn de mètodes.

Per què usar Optional?

- **Evitar NullPointerException:** Redueix errors associats amb l'ús incorrecte de null.
- **Millorar la llegibilitat del codi:** Fa explícit que un valor pot ser absent, eliminant la necessitat de comentaris addicionals.
- **Simplificar el tractament de valors nuls:** Proporciona maneres segures i senzilles de manejar l'absència de valors sense condicions complexes.

En el tema de programació funcional veurem la classe en detall, i quan convé utilitzar-la.

Classe Optional: Creació

Hi ha diverses maneres de crear instàncies de Optional, depenent de si el valor és conegut, pot ser nul, o està absent.

```
// 1. Crear un Optional buit (sense valor)
Optional<String> optBuit = Optional.empty();

// 2. Crear un Optional amb un valor no nul (llança excepció si és nul)
Optional<String> optAmbValor = Optional.of("Hola");

// 3. Crear un Optional que pot tenir valor o estar buit
String possibleNull = null;
Optional<String> optNullable = Optional.ofNullable(possibleNull);
```

Classe Optional: Mètodes bàsics

Optional proporciona diversos mètodes per accedir i gestionar el seu contingut de manera segura:

```
Optional<String> opt = Optional.of("Exemple");

// Comprovar si hi ha valor
boolean teValor = opt.isPresent();
boolean estaBuit = opt.isEmpty();

// Obtenir el valor
String valor = opt.get();  // Compte!! Llança NoSuchElementException si està buit

// Obtenir valor amb alternativa
String valorSegur = opt.orElse("Valor per defecte");
```

Optional és especialment útil en el retorn de mètodes que poden no tenir un resultat vàlid, evitant així la necessitat de retornar null i les comprovacions explícites de null en el codi client.

Classe Objects

La classe **Objects** (java.util.Objects) proporciona **mètodes estàtics d'utilitat** per a operacions comunes sobre objectes. És especialment útil per evitar **NullPointerException** i realitzar operacions segures.

```
// Comprovació de nuls
Objects.isNull(objecte)        // Retorna true si és null
Objects.nonNull(objecte)       // Retorna true si no és null
Objects.requireNonNull(obj)    // Llança NullPointerException si és null

// Comparació d'objectes
Objects.equals(obj1, obj2)     // Compara de forma segura contra null
Objects.deepEquals(obj1, obj2) // Compara arrays i objectes niuats

// Càlcul de hashCode
Objects.hash(obj1, obj2, ...)  // Genera hashCode per múltiples objectes
Objects.hashCode(obj)          // Retorna hashCode o 0 si és null

// toString amb valor per defecte si és null
String text = Objects.toString(objectePotserNull, "Valor per defecte");

// toString que mostra la identitat de l'objecte
String identitat = Objects.toIdentityString(objecte);  // retorna "tipus@hashcode"
```

Altres classes amb mètodes d'utilitat

- **Files**: Proporciona mètodes per a operacions amb fitxers i directoris
- **Paths**: Facilita la creació d'objectes Path per representar rutes
- **Collections**: Ofereix mètodes estàtics per a manipular i transformar col·leccions, com ordenació, cerca, còpia, transformació...