



كلية الهندسة المعلوماتية

برمجة 3

Java Programming

ا.د. علي عمران سليمان

محاضرات الأسبوع الرابع

Interfaces

الفصل الأول 2024-2025

- **Abstract Classes.**
- **Abstract Methods.**
- **Interfaces.**
- **Fields in Interfaces.**
- **Implementing Multiple Interfaces.**
- **Polymorphism with Interfaces.**
- **Default Methods**

## References

- Deitel & Deitel, Java How to Program, Pearson; 10th Ed(2015)

- د.علي سليمان، بني معطيات بلغة JAVA، جامعة تشرين 2013-2014

## Overriding method (replacement or expansion)

• يوجد نوعين لإعادة تعريف الطرائق Overriding  
-هما الاستبدال replacement والتوسيع expansion

• الاستبدال replacement: تطرقنا لإعادة كتابة طريقة `get_salary()` سابقاً، وقلنا إن الكائن من إي صنف هو الذي يحدد الطريقة التي سيناديها والموجوده بنفس الصنف، وفي حال الرغبة بمنادات الطريقة من صنف الأب تسبق `super.get_salary()`؛ واستخدمت طريقة الاستبدال نظراً لأن كل طريقة جديدة هي استبدال للطريقة الموروثة.

• استخدام constructors طريقة التوسيع، (مثال آخر: استخدام طريقة `printAllDatails()` لطباعة معلومات `Person` وتوسيعها ضمن الصنف `Employee` لإكمال ما يخص الموظف هنا أول سطر سيكون مناداة طريقة طباعة `Person` وفق التالي `super.printAllDatails()`؛ ثم كتابة ما يخص الموظف)، عند اشتقاق كائن من الموظف ومناداة طريقة الطباعة سينفذ طريقة الأم ويكمل بتنفيذ طريقة الابن.

## Reference type and object type

- ليكن لدينا مرجع **e1** من نوع `Employee` ويشير لكائن من نوع `Employee`.  
`Employee e1 = new Employee ("adam",30,"Hama", ...);`
- ليكن لدينا مرجع **e2** من نوع `Employee` ويشير لكائن من صنف `SalariedEmployee` (موظف شهري صنف مشتق من الأول). حيث أن لكل منها المعادلة التي تحسب راتبه.  
`Employee e2 = new SalariedEmployee ("mona",20,"Aleppo", ..., ,800,50);`
- عند مناداة الطريقة `e1.get_salary()` سيتم تنفيذ الطريقة الموجوده ضمن `Employee` وبالمناداة `e2.get_salary()` سيتم تنفيذ الطريقة ضمن `SalariedEmployee` أي أن الكائن هو من يحدد أية طريقته سيتم تنفيذها.
- إن `get_salary()` موجوده ضمن الصنف الأب وبالتالي معرفة على كل الأصناف الوارثة له.
- بفرض أن طريقة ما معرفة ضمن `SalariedEmployee` وتم نداؤها من **e2** لن يتم التعرف عليها رغم أن الكائن من نفس الصنف الموجوده به الطريقة،
- إن نوع المرجع **e2** هو `Employee` ولن يتعرف إلا على الطرق التي لها تعريف ضمن الصنف الأساس `Employee`.
- لا يمكن لمرجع من الصنف المشتق `SalariedEmployee` أن يؤشر إلى كائن من الصنف الأساس:
- `SalariedEmployee e3 = new Employee ("adam",30,"Hama", ...); // X`

- لقد تطرقنا لبناء صنف أساس واشتقاق أصناف منه (وراثته) وإمكانية الحصول على كائنات منهم.
- بفرض أننا نحتاج لصنف معمم generalized وهو بمثابة قالب Template ولا نرغب بأن يتم اشتقاق كائن منه بل من أجل أن يصف الصنف ومحتوياته ونجبر كل الوارثين بنمط محدد ويكون بجعله abstract أي بمثابة صنف أساس فقط للأصناف الأخرى، كمثال نحن نملك SalariedEmployee أو HourlyEmployee فقط وبالتالي كل كائن هو أحد النوعين، عندها نحول الصنف Employee إلى صنف مجرد.
- الطريقة المجردة abstract ليس لها جسم ويجب overridden في كل الأصناف الفرعية غيرالمجردة، وغير ذلك سنحصل على خطأ.
- أي صنف يحتوي على طريقة مجردة abstract يصبح تلقائياً صنفاً مجرداً abstract ويتم ذلك بإضافة الكلمة المفتاحية abstract في تعريف الصنف ما قبل الكلمة المفتاحية class وبعد نوع الوصول للصنف وعندها سيمنع المطابق اشتقاق كائن منه ويعطي خطأ عند محاولة ذلك.

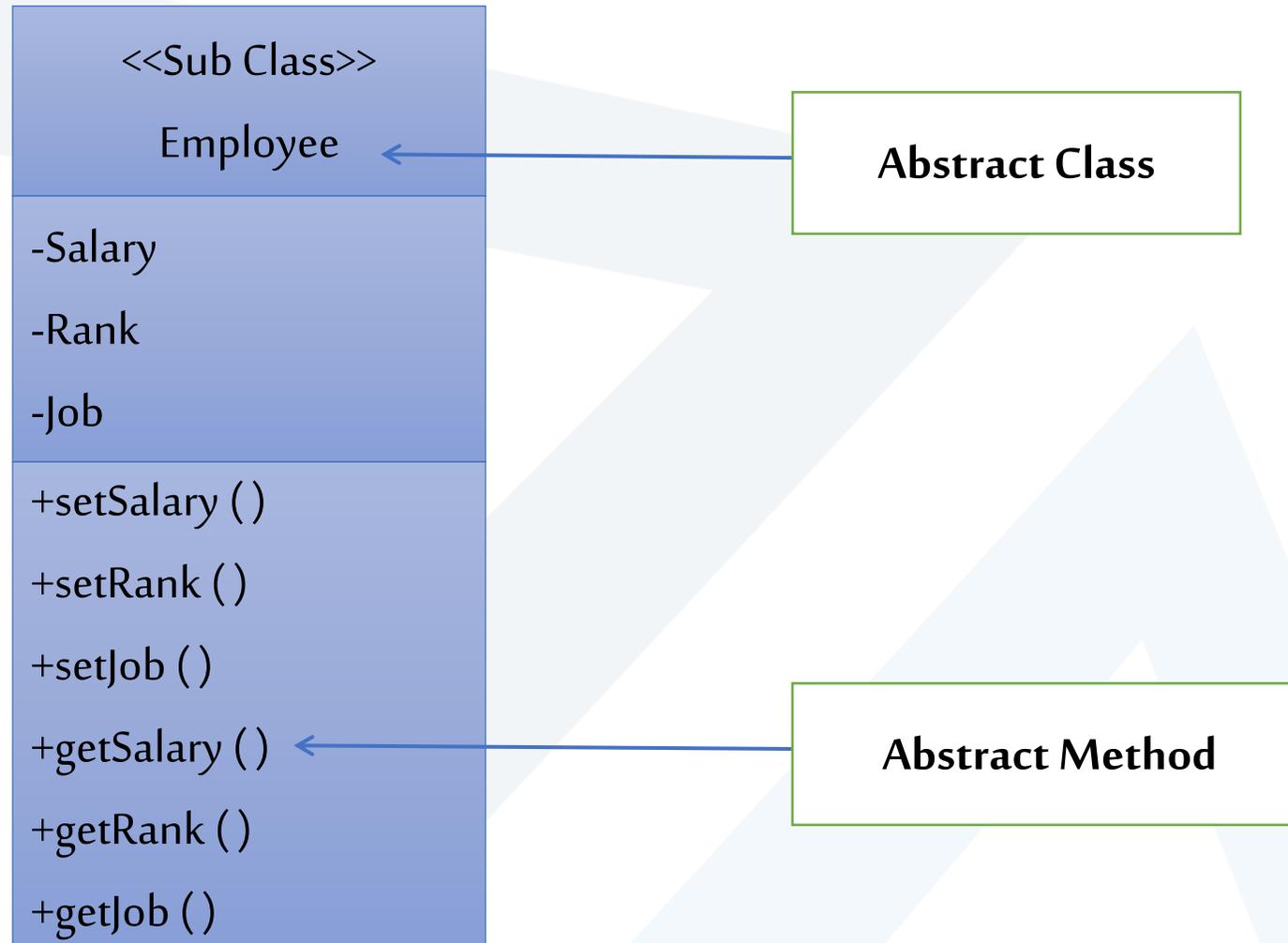
- يمثل الصنف مجرد الشكل العام أو الشكل المجرد لجميع الأصناف المشتقة منه.
- الشكل العام للطريقة المجردة:

`AccessSpecifier abstract ReturenType MethodName(ParameterList);`

`Ex: public abstract double getSalary ( );`

- على كل الأصناف الوارثة لصنف مجرد وغير مجردة أن تقوم بكتابة كل الطرق المجردة الموروثة من الصنف الأساس كل بما ينسجم مع مهمته.
- الطرق التي لن تتغير تكتب ضمن الصنف المعمم، وعدم كتابتها ضمن الأصناف الوارثة لتوفير تكرار الكتابه.
- يتم استخدام طرق مجردة للتأكيد على ضرورة تنفيذها `implements` من قبل الصنف الفرعي.
- إذا فشل الصنف الفرعي في `override` لأي طريقة مجردة ، سينتج خطأ في المترجم.

# Abstract Methods



- لتحويل الطريقة `getSalary()` في الصنف `Employee` إلى طريقة مجردة توضع `abstract` ما قبل القيمة المعادة وحذف جسمها،
- سيحصل خطأ يطالب بتحويل الصنف `Employee` المدروس سابقاً لصنف مجرد ويتم بوضع `abstract` مابعد نوع الوصول والكلمة المفتاحية `class` ستلاحظ عدم إمانية اشتقاق كائن منه.
- عند اشتقاق صنف غير مجرد `SalariedEmployee` من الصنف المجرد `Employee` ستجد وجود خطأ يطالب بإعادة كتابة الطريقة `getSalary()` لأنها طريقة مجردة.
- هذا ينطبق على الصنف `HourlyEmployee` أيضاً، وكل الأصناف المشتقة من الصنف المجرد `Employee`.
- إذا كانت كل طرائق الصنف مجردة تحول إلى واجهه `Interfaces` وهذا ماسنهتم به الآن.
- الشكل العام لتعريف الواجهة:

```
public interface InterfaceName  
{ (Prototype Method headers...) }
```

- الواجهة interface تعميم للاصناف المجردة وتحتوي الطرق المجردة فقط.
- الغاية من الواجهة interface هو تحديد السلوكيات للاصناف الأخرى المحققة لها.
- الواجهة interface. هي عبارة عن مجموعة من تصريحات الطرائق بدون أي أجسام لطرائقها، أي أن طرائق الواجهة يكتب نموذجها فقط proto type (أي مجرد تو اقيع طرائق).
- إن هذا التحديد بدوره، يفرض من قبل المترجم أو نظام وقت التنفيذ، والذي يتطلب أن تكون أنواع البارامترات التي تمرر عادة إلى الطرائق تتوافق بصرامة مع النوع المحدد في الواجهة.
- عندما يقوم صنف بتحقيق أو بناء implements واجهة، فيجب أن يبني جميع الطرائق المصرح عنها فيها، بهذه الاسلوب، فإن الواجهات تتطلب وجود صنف بناء يملك طرائقها بنفس التواقيع المحددة.
- يقال غالبًا أن الواجهة تشبه "العقد" contract، وعندما تنفذ صنف لواجهة، يجب أن يلتزم بالعقد.

## Contract

```

Class Photograph
public String description()
    { return descript; }
    .
    .
    .
    .
public int Price()
    {return price;}
public int lowestPrice()
    {return price/2;}
    
```



```

interface Sellable
1 - description()
    .
    .
    .
    .
    .
    .
    .
    .
7 - listPrice()
8 - lowestPrice()
    
```

**Class Photograph**

**Implements interface Sellable**

**Interface**

## interface Sellable

```
/** Interface for objects that can be sold. */  
public interface Sellable {  
  
    /** description of the object */  
    public String description();  
  
    /** list price in cents */  
    public int listPrice();  
  
    /** lowest price in cents we will accept */  
    public int lowestPrice();  
}
```

الواجهة Sellable

## class Photograph implements Sellable

```
/** Class for photographs that can be sold */
public class Photograph implements Sellable {
    private String descript; // description of this photo
    private int price; // price we are setting
    private boolean color; // true if photo is in color
    public Photograph(String desc, int p, boolean c) { // constructor
        descript = desc; price = p; color = c; }
    public String description() { return descript; }
    public int listPrice() { return price; }
    public int lowestPrice() { return price/2; }
    public boolean isColor() { return color; }
    public String toString() { // for printing
        return "( descript: " + descript + " SlistPrice: " + price + ", lowestPrice: " + price/2 + ", Color: " + color + ")";
    }
}
```

الصنف Photograph يحقق الواجهه Sellable

• لنفرض أننا نرغب بإنشاء جرد بالتحف التي نملكها، مصنفة كأغراض من أنواع مختلفة. وقد نرغب بتعريف بعض الأشياء على أنها قابلة للبيع، بحيث إننا قمنا ببناء الواجهة Sellable وبعدها **بناء الصنف Photograph** الذي يبني الواجهة Sellable والمحقق لكل طرق الواجهه Sellable بحسب الحاجة، للإشارة إلى أننا نرغب بأن نكون قادرين على بيع أي من الأغراض من الصنف Photograph، **بالإضافة إلى أنه تمت إضافة الطريقة isColor الخاصة بالأغراض من النوع Photograph.**

• لا يمكن إنشاء كائن من الواجهه. `Sellable sel1= new Sellable(); // ERROR!`

• يمكن إنشاء كائن من الصنف المحقق للواجهه وتخزين عنوانه في مرجع من نمط الواجهه.

`Sellable sel1= new Photograph();` وهنا يتم تحقيق مبدأ Polymorphism.

• تكون الوراثة المتعددة في لغة الجافا متاحة من أجل الواجهات وليس من أجل الأصناف (واجهه ترث عدة واجهات).

• إن طرائق واجهة ما ليس لها أجسام تعريف وبالتالي لن يحصل التباس عند وجود طريقتين بنفس التوقيع في واجهتين نظراً لعدم وجود اجساد لها في الواجهه المورثة.

- تحتوي جميع الكائنات في Java على طريقة خاصة تسمى toString والتي تعرض تمثيل الشريط المحرفي String لمحتويات الكائن.

- الطريقة toString موجوده ضمن الكائن Object وتسمى هذه الطرق بطرق الخدمات العامة أيضًا، أو الواجهة العامة التي يوفرها الصنف لعملائه.

- عندما يتم ربط كائن بسلسلة، يتم استدعاء طريقة الكائن toString ضمناً للحصول على تمثيل سلسلة للكائن.

```
Photograph Pho1 = new Photograph("adam", 100, true);
```

```
System.out.println(Pho1);
```

- يمكن أيضًا استدعاء طريقة toString بشكل صريح.

```
System.out.println(Pho1.toString());
```

- يجب أن تتم إعادة كتابتها overridden لتعطي المطلوب كمثال :

```
public String toString() { // for printing
    return "(descript: " + descript + " SlistPrice: " + price + ", lowestPrice: " + price/2 + ", Color: " + color + ")";
```

وفي حال عدم كتابتها ستعطي Pho1 @1fee6fc اسم الكائن وعنوانه ضمن الذاكرة.

يمكن أن تضم مجموعة أغراض ونوعاً آخر من الأغراض التي يمكن نقلها، من أجل هذه الأغراض، يمكن أن نعرف الواجهة التالية:

```
/** Interface for objects that can be transported. */
```

```
public interface Transportable
```

```
{
```

```
/** weight in grams */
```

```
public int weight();
```

```
/** whether the object is hazardous */
```

```
public boolean isHazardous();
```

```
}
```

الواجهة `Transportable`: يمكن أن نعرف الصنف `BoxedItem` من أجل التحف المتنوعة التي يمكن أن نبيعها، نحزمها أو نشحنها. وبالتالي، يبني الصنف `BoxedItem` طرائق الواجهة `Sellable` والواجهة `Transportable` مع إضافة طرائق مخصصة لتحديد قيمة التأمين لشحن صندوق وأبعاد الصندوق المشحون.

## BoxedItem implements Sellable, Transportable

```
/** Class for objects that can be sold, packed, and shipped. */  
public class BoxedItem implements Sellable, Transportable  
{ private String descript;           // description of this item  
  private int price;                 // list price in cents  
  private int weight;                // weight in grams  
  private boolean haz;              // true if object is hazardous  
  private int height;               // box height in centimeters  
  private int width=0;              // box width in centimeters  
  private int depth=0;              // box depth in centimeters  
  public BoxedItem( String desc,int p, int w, boolean h)  
    {descript= desc; price= p; weight= w; haz= h;} //Constructor  
  public String description()        { return descript; }  
  public int listPrice()             { return price; }  
  public int lowestPrice()           { return price/2; }  
  public int weight()                { return weight; }  
  public boolean isHazardous()       { return haz; }  
  public int insuredValue()          { return price*2; }
```

## BoxedItem implements Sellable, Transportable and main()

```

public void setBox(int h, int w, int d)
{ height = h; width = w; depth = d; } // end setBox
public String toString() { // for printing BoxedItem
return "descript: " + descript + " SlistPrice: " + price + ", lowestPrice: " + price/2 + ",\n weight: " + weight + ", isHazardous: " + haz + ",
insuredValue: " + price*2 + ",\n height: " + height + ", width: " + width + ", depth: " + depth + ")); } //end toString in BoxedItem
public static void main( String args[] )
{ Photograph Pho1 = new Photograph("adam", 100, true );
System.out.println(Pho1);
System.out.println();System.out.println();
BoxedItem box1 = new BoxedItem("adam", 100,10, true );
System.out.println("\n");
box1.setBox(3, 5, 7); System.out.println("\n\n");
System.out.println(box1.toString());
} // end main
} // end class BoxedItem

```

- عندما ينفذ صنف واجهات متعددة، يجب أن توفر الطرق المحددة من قبل كل منهم.
- لتحديد واجهات متعددة في تعريف تحقيق صنف، تسرد أسماء الواجهات ، مفصولة عن بعضها بعضاً بفواصل، بعد الكلمة المفتاحية `implements`.
- بالعودة إلى مثال التحف، يمكن أن نعرف واجهة للأغراض المؤمن عليه بالعودة إلى مثال التحف، يمكن أن نعرف واجهة للأغراض المؤمن عليها والوارثة لعدة واجهات تذكر مفصولة عن بعضها بعضاً بفواصل ، بعد الكلمة المفتاحية `extends`.  
كما يلي:

```
public interface InsurableItem extends Transportable, Sellable
{
    /** Returns insured Value in cents */
    public int insuredValue();
}
```

- تدمج هذه الواجهة طرائق الواجهة `Transportable` مع طرائق الواجهة `Sellable` وتضيف طريقة أخرى `insuredValue()`

• إن مثل هذه الواجهة يمكن أن تتيح لنا تعريف الصنف `BoxedItem2` كما يلي:

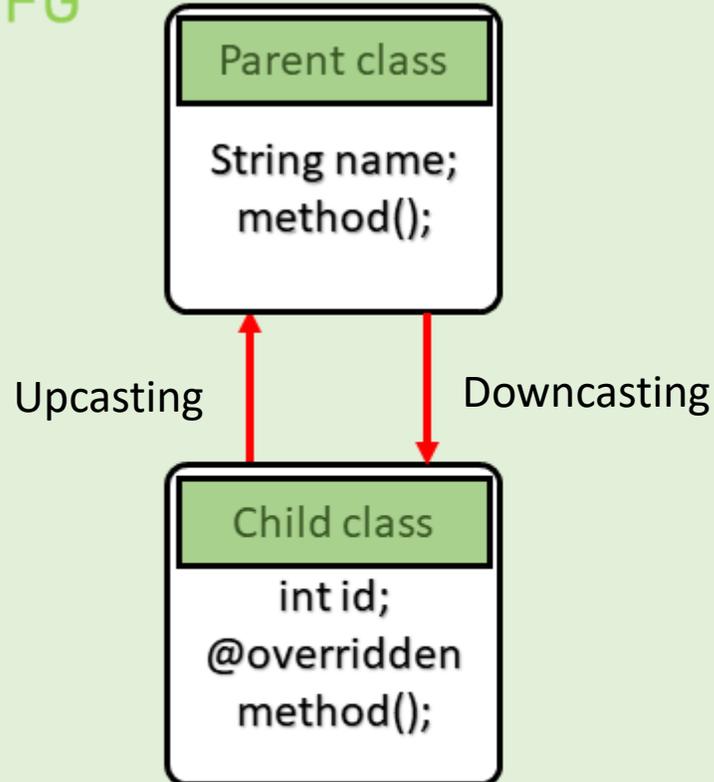
```
public class BoxedItem2 implements InsurableItem { /* ... same code as class BoxedItem */ }
```

- في هذه الحالة، لاحظ أن الطريقة `insuredValue` ليست اختيارية، في حين أنها كانت اختيارية في النسخة السابقة للصنف `BoxedItem`، لأنها معرفه ضمن الواجهه الورثه `InsurableItem`، وكذلك كل الطرق من الواجهتين الموروثتين.
- تتيح لنا الواجهات إرغام الأغراض على بناء طرائق محددة، إلا أن استخدام متحولات الواجهات مع أغراض حقيقية يتطلب أحيانا استخدام تحويل الأنماط.
- تحويل التوسيع `widening` أو `UpCasting` ويحصل بشكل تلقائي أو اوتوماتيكي عند ما يشير مؤشر من نمط الابن إلى كائن من نمط الاب .

- تحويل التضيق `Narrowing` أو `DowenCasting` هي ضرورية في حال مؤشر من نمط الاب إلى كائن من نمط الابن (في حالة `is – like` لأن الصنف الجديد يضيف حقول وطرق غير موجوده عند المورث وبالتالي لايمكن للمؤشر من نوع الاب المسند له غرض من نوع الابن أن يرى ما تمت إضافته) وهنا يتطلب تحويلاً صريحاً، ويجب التأكد من إمكانية القسر قبل اجراء القسر ويوجد عامل مقارنة النوع يختبر ذلك `instanceof` وعندما يكون الجواب `true` نقوم بالقسر الآمن (`object`) `instanceof (type)` وتعرف بـ `run – time type identification (RTTI)`



GFG



```
Parent p = new Child();//Upcasting
p.name = "GeeksforGeeks";
//p.id = 1; Not accessible
p.method();//overridden method
//Trying to Downcasting Implicitly
//Child c = new Parent();
//compile-time error
Child c = (Child)p;//Downcasting Explicitly
c.id = 1;
c.method();
```

MainDemo class

Figure to illustrate the Concept of Upcasting Vs Downcasting

```
//Java program to demonstrate Upcasting Vs Downcasting Parent class
class Parent {String name;
// A method which prints the signature of the parent class
void method(){System.out.println("Method from Parent");} } // Child class
class Child extends Parent {int id;
// Overriding the parent method to print the signature of the child class
@Override void method()
{System.out.println("Method from Child");} }
// Demo class to see the difference between upcasting and downcasting
public class GFG {/* Driver code */ public static void main(String[] args)
{/* Upcasting */Parent p = new Child(); p.name = "GeeksforGeeks";
/*Printing the parentclass name */ System.out.println(p.name);
/* parent class method is overridden method hence this will be executed */
p.method();
// Trying to Downcasting Implicitly Child c = new Parent(); - > compile time error
// Downcasting Explicitly
Child c = (Child)p; c.id = 1;
System.out.println(c.name);System.out.println(c.id);c.method();} }
```

- بفرض أننا صرحنا عن الواجهة `Person` المبينة في المقطع البرمجي التالي. إن الطريقة `EqualTo` للواجهة `Person` تأخذ بارامتراً واحداً من النوع `Person`. وبالتالي، يمكن أن نمرر غرض من أي صنف يبني الواجهة `Person` إليها.

```
public interface Person {
    public boolean equalTo (Person other);    // is this the same person?
    public String getName();                //get this person's name
    public int getAge();                     /* get this person's age */ }

```

- عندما يشير متغير واجهة إلى كائن:

- فقط الطرق المعلنه في الواجهة تكون متاحة، بما يحاكي المعروض في الشريحة الرابعة.
- مطلوب تحويل قسري `casting` للنوع الصحيح للوصول إلى الطرق الأخرى للكائن المشار إليه بواسطة مرجع الواجهة.

- نبين في المقطع البرمجي التالي صنفاً `Student` يبني الواجهة `Person`. تفترض الطريقة `equalTo` أن الوسيط (المصرح عنه من النوع `Person`) هو أيضاً من النوع `Student` ويقوم بتحويل تضيق من النمط `Person` (الواجهة) إلى النمط `Student` (الصنف) باستخدام تحويل صريح. إن عملية التحويل مسموحة في هذه الحالة.

```

public class Student implements Person {
String id; String name; int age;           // simple constructor
public Student (String i, String n, int a){id=i; name=n; age=a;}

// protected int studyHours() {return age/2;}
public String getID () {return id;}        //ID of the student
public String getName(){return name;}     //Person interface
public int getAge() {return age;}        // Person interface
public boolean equalTo (Person other) {
//Person interface cast Person to Student
Student otherStudent = (Student) other; return (id.equals (otherStudent.getID())); }

public String toString(){ return "Student(ID: " + id + ", Name: " + name + ", Age: " + age + ")";
} // end toString
} // end class Student

```

بناء الصنف Student

وبسبب الافتراض الذي قمنا به في بناء الطريقة `equalTo`، يجب علينا أن نتحقق أن تطبيقاً يستخدم أغراضاً من النوع `Student` لن تحاول إجراء مقارنة بين أغراض مع أنواع أخرى من الأغراض. وإلا، فإن تحويل الأنماط في الطريقة `equalTo` سيفشل.

إن القدرة على إجراء تحويلات تضيق من أنماط واجهات إلى أنماط اصناف تتيح لنا كتابة أنواع عامة من بني المعطيات التي تتضمن افتراضات دنيا حول العناصر التي تقوم بتخزينها. نبين في المقطع التالي، كيف نقوم ببناء مجلد يخزن أزواجاً من الأغراض التي تبني الواجهة `Person`، تقوم الطريقة `remove` بعملية بحث ضمن محتويات المجلد وتحذف الزوج المحدد، إذا كان موجوداً، وكما في الطريقة `findOther` فإنها تستخدم الطريقة `equalTo` للقيام بذلك.

```
public class PersonPairDirectory
{
    // ... instance variables would go here ...
    public PersonPairDirectory()
        { /* default constructor goes here */}
    public void insert (Person person, Person other)
        { /* insert code goes here */}
    public Person findOther (Person person)
        {return null; } // stub for find
    public void remove (Person person, Person other)
        { /* remove code goes here */}
} // end class PersonPairDirectory
```

مثال عن صنف

الآن، بفرض أننا ملأنا المجلد myDirectory، بأزواج من الأغراض Student التي تمثل أزواجاً متجاورة. من أجل إيجاد مجاور غرض smart\_one من النوع Student، يمكن أن نحاول القيام بما يلي (وهو خاطئ):

```
Student cute_one=myDirectory.findOther (smart_one); // wrong!
```

إن الأمر السابق يسبب خطأ ترجمة يدعى "explicit-cast-required"، المشكلة هنا هي أننا نحاول القيام بتحويل تضيق بدون معامل تحويل صريح. بمعنى، أن القيمة المعادة من الطريقة findOther هي من النوع Person في حين أن المتحول cute\_one التي يتم إسنادها إليه هو من النوع الأضيق Student، وهو صنف يبني الواجهة Person. وبالتالي، يجب استخدام عملية تحويل صريحة لتحويل النمط Person إلى النمط Student، كما يلي:

```
Student cute_one=(Student)myDirectory.findOther(smart_one);
```

إن تحويل القيمة من النوع Person المعادة من الطريقة findOther إلى النمط Student تعمل بشكل جيد طالما أننا متأكدين من أن الاستدعاء لـ myDirectory.findOther يعطينا حقيقة غرضاً من النوع Student. عموماً، يمكن أن تكون الواجهات أداة قيمة لتصميم بني معطيات عامة، والتي يمكن تخصيصها من قبل مبرمجين آخرين من خلال استخدام تحويل الأنماط.

- في إطار عمل التعميمات generics framework لاستخدام الأنماط المجردة بطريقة تتجنب العديد من التحويلات الصريحة.
- النوع العمومي generic type هو نوع لا يعرف في زمن الترجمة، وإنما يصبح محدداً كلياً في زمن التنفيذ.
- يتيح لنا إطار عمل التعميمات تعريف الصنف بدلالة مجموعة من البارامترات ذات الأنواع الشكلية formal type parameters التي يمكن أن تستخدم، على سبيل المثال، لتجريد أنماط بعض المتحولات الداخلية للصنف، تستخدم أقواس زاوية angle brackets لحصر قائمة البارامترات ذات الأنماط الشكلية، على الرغم من أن أي معرف صالح يمكن أن يستخدم من أجل بارامتر ذو نمط شكلي.
- إذا كان لدينا صنف قد عرف مع هذه البارامترات، عندها يمكن أن نقوم بتهيئة أو إنشاء غرض من هذا الصنف يستخدم بارامترات ذات أنماط فعلية للإشارة إلى الأنماط الحقيقية المستخدمة.
- يبين المقطع البرمجي التالي الصنف Pair الذي يخزن أزواجاً قيمة-مفتاح key-value pairs، حيث إن أنواع القيمة والمفتاح محددة بالبارامترات V و K على التوالي. تقوم الطريقة main بإنشاء مثلين من هذا الصنف، الأول من أجل زوج String-Integer والثاني Student-Double.

## interface Sellable

```
public class Pair<K, V> { K key; V value;  
public void set(K k, V v){key= k;value= v;}  
public K getKey() { return key; }  
public V getValue() { return value; }  
public String toString() { return "[" + getKey() + ", " + getValue() + "]; }  
}
```

```
public static void main (String[] args)  
{  
    Pair<String,Integer> pair1=new Pair<String,Integer>();  
    pair1.set(new String("height"), new Integer(36));  
    System.out.println(pair1);  
    Pair<Student,Double> pair2=new Pair<Student,Double>();  
    pair2.set(new Student("A5976","Sue",19),new Double(9.5));  
    System.out.println(pair2);  
    System.out.println("\n\n"+pair2.equals(pair1));  
    } // end main  
} // class Pair
```

[height, 36]

[Student(ID: A5976, Name: Sue, Age: 19), 9.5]

false

- كما ذكرنا سابقاً `Sellable pho= new Sellable (); // wrong!` هي عملية غير مسموحة.
- تسمح Java بإنشاء متغيرات مرجعية للواجهة تشير إلى كائنات من الأصناف المحققة لها.
- يمكن للمتغير المرجعي للواجهة أن يشير إلى أي كائن يقوم بتنفيذ تلك الواجهة، بغض النظر عن نوع صنفه.
- في كود المثال ، تم التصريح عن متغيرين مرجعيين ، هما `pho1, pho2` .
- يشير المتغير المرجعي `pho1` إلى كائن `Photograph` ويشير متغير `pho2` إلى كائن `BoxedItem`.
- عندما ينفذ صنف واجهة ، يتم إنشاء علاقة وراثية تعرف باسم وراثية الواجهة.

```
Sellable pho1= new Photograph ();
```

```
Sellable pho2= new BoxedItem ();
```

• عند ذكر `pho1` ستظهر كل الطرق المتاحة والمعرفة ضمن الواجهة المحققة.

• الطرق المعرفة ضمن الصنف المحقق للواجهة لن يتم التعرف عليها لعدم معرفة الواجهة بها مثل `pho1.isColor();` على غرار مرجع من نوع ويشير على كائن من نوع مشق لن يصل للطرق التي لم تعرف في الأصل الشريحة أربعة.

- يمكن أن تحتوي الواجهة على حقول تصريحات:
  - يتم التعامل مع جميع الحقول في الواجهة على أنها `final and static`.
  - لأنها تصبح نهائية بشكل تلقائي، يجب توفير قيمة تهيئة لها.

```
public interface Doable
{
    int FIELD1 = 1, FIELD2 = 2;
    (Method headers...)
}
```

- في هذه الواجهة ، `FIELD1` و `FIELD2` هما متغيران صحيحان نهائيان ثابتان `final and static`.
- أي صنف ينفذ هذه الواجهة لديها حق الوصول إلى هذه المتغيرات.

# انتهت محاضرات الأسبوع الرابع

العنوان	رقم الفقرة
استخدام مربعات الحوار: المدخلات والمخرجات الأساسية مع مربعات الحوار	3.9
إنشاء رسومات بسيطة عرض الخطوط ورسمها على الشاشة	4.14
رسم المستطيلات والأشكال البيضاوية استخدام الأشكال لتمثيل البيانات	5.10
الألوان والأشكال المعبأة رسم قوس قزح ورسومات عشوائية	6.13
رسم الأقواس رسم الشكل الحلزوني بالأقواس	7.13
استخدام الكائنات مع الرسومات تخزين الأشكال ككائنات	8.18
عرض النص والصور باستخدام الملصقات توفير معلومات الحالة	9.8
الرسم باستخدام تعدد الأشكال: تحديد أوجه التشابه بين الأشكال	10.8
توسيع الواجهة: استخدام مكونات واجهة المستخدم الرسومية ومعالجة الأحداث	14.17

- ▶ تحتوي حزمة `javax.swing` على العديد من الأصناف التي تساعدك على إنشاء واجهات مستخدم رسومية GUIs
- ▶ تسهل مكونات واجهة المستخدم الرسومية إدخال البيانات من قبل مستخدم البرنامج وعرض المخرجات للمستخدم.
- ✓ `showInputDialog()` و `showMessageDialog()` من الصنف `JOptionPane` | مناهج `static`.
- ✓ غالبًا ما تحدد مثل هذه المناهج المهام المستخدمة بشكل متكرر أو خدماتي.
- ✓ يتم استدعاؤها عادةً باستخدام اسم فئة المنهج متبوعًا بنقطة (.) واسم الطريقة و  
والوسطاء ما بين قوسين، كما يلي:

*ClassName.methodName( arguments )*

- ✓ لاحظ أنك لم تقم بإنشاء كائن من الصنف `JOptionPane` لاستخدام `static method` `showMessageDialog()`

- ▶ المنهج `showMessageDialog()` من الصنف `JOptionPane` من الحزمة `javax.swing` له الشكل العام التالي.
- ▶ `showMessageDialog(arg1,arg2)` يعرض مربع حوار العرض نافذة ويتطلب بارامترين،
  - الأول `arg1` يساعد تطبيق `Java` على تحديد مكان وضع مربع الحوار. إذا كانت الوسيطة الأولى `null`، فسيتم عرض مربع الحوار في وسط الشاشة.
  - الوسيط الثاني `arg2` هو السلسلة `String` المراد عرضها في مربع الحوار.
  - وتظهر نافذة فارغة وفق الحالة الافتراضية.

✓ يسمح صندوق حوار الإدخال `input-dialog` للمستخدم بإدخال البيانات في البرنامج.

✓ يعرض المنهج `showInputDialog()` من الصنف `JOptionPane` مربع حوار الإدخال.

✓ يحتوي على مؤشر الإدخال في حقل يُعرف بحقل النص `text field` حيث يمكن للمستخدم إدخال نص فيه.

✓ المنهج `showInputDialog()` يقوم بإرجاع مرجع سلسلة `String` يحتوي على عنوان الأحرف التي كتبها المستخدم في حقل النص، وحتى الأرقام يعبر عنها كحروف.

✓ إذا تم الضغط على زر إلغاء `Cancel` في مربع الحوار أو ضغط على مفتاح `Esc key` على لوحة المفاتيح، فإن الطريقة ترجع `null` للدلالة على تجاهل الإدخال.

✓ يقوم المنهج `showMessageDialog()` بعرض نص منسقاً قد يكون بـ `String.format()`، كما يعمل المنهج

`System.out.printf` بإرجاع سلسلة منسقة وعرضها في نافذة الأوامر. باستثناء أن `String.format()` يُرجع عنوان السلسلة المنسقة إلى مرجع من النمط `String`.

# Dialog Boxes

## 3.10 GUI & Graphics

```
1 // Fig. 3.17: Dialog1.java
2 // Printing multiple lines in dialog box.
3 import javax.swing.JOptionPane; // import class JOptionPane
4
5 public class Dialog1
6 {
7     public static void main( String args[] )
8     {
9         // display a dialog with the message
10        JOptionPane.showMessageDialog( null,
11                                       "Welcome\nto\nJava" );
12    } // end main
13 } // end class Dialog1
```



# Using Dialog Boxes

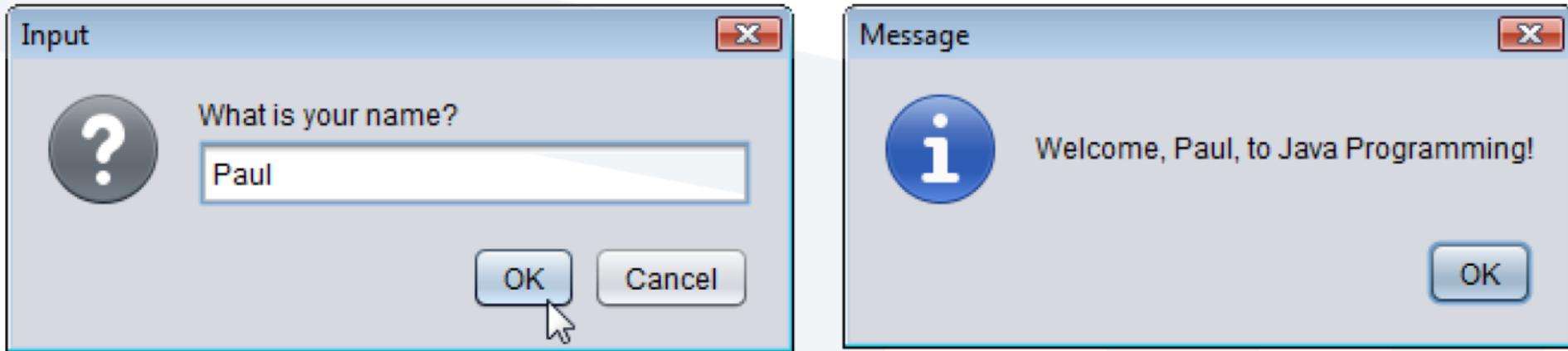


## 3.10 GUI & Graphics

```
1 // Fig. 3.18: NameDialog.java
2 // Basic input with a dialog box.
3 import javax.swing.JOptionPane;
4 public class NameDialog
5 {
6 public static void main( String args[] )
7 {
8 // prompt user to enter name
9 String name = JOptionPane.showInputDialog( "What is your name?" );
10 // create the message
11 String message = String.format( "Welcome, %,s, to Java Programming!", name );
12 // display the message to welcome the user by name
13 JOptionPane.showMessageDialog( null, message );
14 } // end main
15 } // end class NameDialog
```

Displays an input Dialog to obtain data from user

Creates a formatted String containing the user entered in the user entered in the input dialog



**Fig. 3.18** | Obtaining user input from a dialog. (Part 2 of 2.)

نظرًا لأن الطريقة `showInputDialog()` تُرجع مرجع سلسلة نصية، إذا كان المدخل عدد سيعامل كنص ويجب تحويله إلى عدد لاستخدامها في العمليات الحسابية.

الطريقة `Integer.parseInt(args1)` من الصنف `Integer` من الحزمة `(package java.lang)`، حيث الوسيط `args1` هو سلسلة تمثل عددًا صحيحًا وترجع القيمة كعدد نمط `int`.

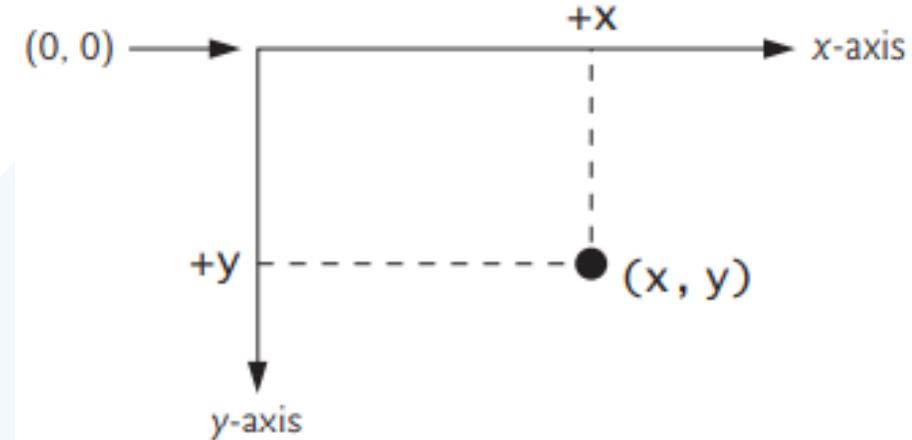
▶ القسم الأول الرسومات في الجافا Graphics Java في هذا القسم نتعرف على الأدوات التي توفرها لغة جافا للرسم والتلوين على شاشة البرنامج.

▶ القسم الثاني واجهات المستخدم الرسومية (Graphical User Interface . GUI) في هذا القسم سنتحدث عن مجموعة من أدوات لغة جافا الخاصة بتصميم واجهات المستخدم أو شاشات البرنامج، والتي تساعد المستخدم على التفاعل مع البرنامج بصورة أسهل وأبسط ولا تتطلب المعرفة الدقيقة بالبرمجة ولغاتها.

العنوان	رقم الفقرة
استخدام مربعات الحوار: المدخلات والمخرجات الأساسية مع مربعات الحوار	3.9
إنشاء رسومات بسيطة عرض الخطوط ورسمها على الشاشة	4.14
رسم المستطيلات والأشكال البيضاوية استخدام الأشكال لتمثيل البيانات	5.10
الألوان والأشكال المعبأة رسم قوس قذح ورسومات عشوائية	6.13
رسم الأقواس رسم الحلزونات بالأقواس	7.13
استخدام الكائنات مع الرسومات تخزين الأشكال ككائنات	8.18
عرض النص والصور باستخدام الملصقات توفير معلومات الحالة	9.8
الرسم باستخدام تعدد الأشكال: تحديد أوجه التشابه بين الأشكال التمرين	10.8
توسيع الواجهة: استخدام مكونات واجهة المستخدم الرسومية ومعالجة الأحداث	14.17

- ✓ نظام الإحداثيات في Java عبارة عن مخطط scheme لتحديد النقاط على الشاشة.
- ✓ الزاوية العلوية اليسارية من نافذة واجهة المستخدم الرسومية لها الإحداثيات (0,0).
- ✓ يتكون زوج الإحداثيات من **x-coordinate** (الإحداثي الأفقي **horizontal coordinate**) وإحداثي **y-coordinate** (إحداثي رأسي **vertical coordinate**).

- إحداثي  $x$  هو الموقع الأفقي الذي يتم التحرك عليه من اليسار إلى اليمين.
- إحداثي  $y$  هو الموقع الرأسي الذي يتم التحرك عليه من أعلى إلى أسفل.
- تشير الإحداثيات إلى مكان عرض الرسومات على الشاشة.
- يصف المحور  $x$  كل الإحداثيات الأفقية ، ويصف المحور  $y$  كل الإحداثيات الرأسية.
- تقاس وحدات التنسيق بالبكسل. يشير المصطلح المعروف بكسل إلى "عنصر الصورة"، وهو أصغر وحدة دقة قياس في شاشة العرض.



نستخدم فئتين لبرمجة واجهة المستخدم وهما `java.awt` و `javax.swing`.

- قدمت لغة الجافا برمجة واجهات المستخدم لأول مرة وكانت كل الأصناف موجودة في مكتبة تسمى `awt` (Abstract Window Toolkit) والتي تقوم بضبط اعدادات البرنامج تلقائيا حسب المنصة التي يتم تشغيل البرنامج عليها، وهي تنفع في بناء واجهات مستخدم بسيطة، ولكن لا تجدي نفعا في بناء واجهات مستخدم محترفة ومتميزة.
- تم استبدال حزمة `awt` بحزمة اكثر تميزا وكفاءة هي فئة `swing` وعرفت ب `light components weight` اي المكونات الخفيفة لأنها تعتمد علي انشاء الكائنات من دون الإعتماد علي منصة التشغيل على خلاف `awt` والتي تعرف `heavy components weight` المكونات الثقيلة لأنها تعتمد وتتعامل مع منصة التشغيل.
- ومن اجل التفريق بين اصناف حزمة `awt` واصناف حزمة `swing` يتم اضافة السابقة قبل اسم كل صنف من اصناف فئة `swing`.
- طريقة تصميم وترتيب ال `GUI Application Program Interface` تعتبر من افضل الأمثلة علي استخدام الوراثة والأصناف والواجهات `.Interfaces`.

كل برنامج رسومي يستخدم نافذة إطار window frame أو أكثر ولكل نافذة إطار شريط عنوان titel bar وحدود border لكي يظهر الإطار نستخدم الصنف JFream من الحزمة javax.swing ويجب:

1- إنشاء كائن من JFream وفق

2- تجديد مقاس الإطار من الطريقة setSize .

3- إضافة الرسمة أو ماتم تجميعها ونرغب بعرضه إلى الإطار

4- جعل الإطار مرئي تستخدم الطريقة show لجعل مدير عرض النافذة window manager يعرضها افتراضيا هي false.

```
appli.setVisible( true );
```

5- عند تنفيذ البرنامج يتم إظهار الإطار وينتهي تنفيذ main ولكن يظل البرنامج يعمل والإطار ظاهر على الشاشة ويمكن تحريكه وتغيير حجمه و ... ، وعند إغلاق نافذة الإطار بالضغط على أيقونة الإغلاق من شريط العنوانه يظل البرنامج يعمل ولا يحدث شيء سوى إختفاء الإطار، ومن أجل إنهاء البرنامج يجب استخدام System.exit(0) والتي يجب أن تكون بنهاية main ولكن تخلق مشكلة جديدة وهي ظهور النافذة للحظة وجيزة وينتهي فوراً والرغبة هي إنهاء البرنامج عند الضغط المستخدم على أيقونة الغلق في

شريط العنوان وهنا نجد اسهل طريقة استخدام المنهج : appli.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

أو معالجة حدث النقر على أيقونة الغلق من أجل إنهاء البرنامج إضافة على إغلاق النافذة

# Creating Simple Drawings



## 4.17 GUI & Graphics

```
1 // Fig. 4.18: DrawPanel.java
2 // Using drawLine to connect the corners of a panel.
3 import java.awt.Graphics;
4 import javax.swing.JPanel;
5
6 public class DrawPanel extends JPanel
7 {
8     // draws an X from the corners of the panel
9     public void paintComponent( Graphics g )
10    {
11        // call paintComponent to ensure the panel displays correctly
12        super.paintComponent( g );
13
14        int width = getWidth(); // total width
15        int height = getHeight(); // total height
16
17        // draw a line from the upper-left to the lower-right
18        g.drawLine( 0, 0, width, height );
19
20        // draw a line from the lower-left to the upper-right
21        g.drawLine( 0, height, width, 0 );
22    } // end method paintComponent
23 } // end class DrawPanel
```

Import the classes `Graphics` and `JPanel` for use in this source code file.

`DrawPanel` inherits the existing capabilities of class `JPanel`

`paintComponent` *must* be displayed as shown here

This should be the first statement in method `paintComponent`

Determines the width and height of the `DrawPanel` with inherited methods

Draws a line from the top-left to the bottom-right of the `DrawPanel`

Draws a line from the bottom-left to the top-right of the `DrawPanel`

Deitel & Deitel (C) 2010 Pearson Education, Inc. All rights reserved.

**Fig. 4.18** | Using `drawLine` to connect the corners of a panel.

```
1 // Fig. 4.19: DrawPanelTest.java
2 // Application to display a DrawPanel.
3 import javax.swing.JFrame;
4
5 public class DrawPanelTest
6 {
7     public static void main( String[] args )
8     {
9         // create a panel that contains our drawing
10        DrawPanel panel = new DrawPanel();
11
12        // create a new frame to hold the panel
13        JFrame application = new JFrame();
14
15        // set the frame to exit when it is closed
16        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
17
18        application.add( panel ); // add the panel to the frame
19        application.setSize( 250, 250 ); // set the size of the frame
20        application.setVisible( true ); // make the frame visible
21    } // end main
22 } // end class DrawPanelTest
```

Imports class JFrame for use in this source code file

Creates a JFrame in which the DrawPanel will be displayed

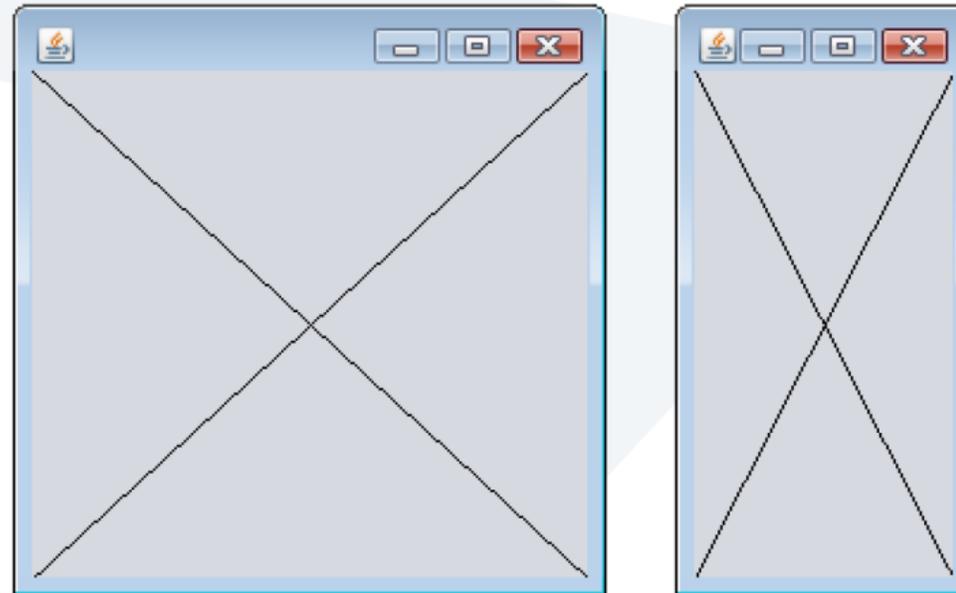
Terminate application when window closes

Attach the DrawPanel to the JFrame

Sets the size of the JFrame

Displays the JFrame on the screen

**Fig. 4.19** | Creating JFrame to display DrawPanel. (Part I of 2.)



**Fig. 4.19** | Creating JFrame to display DrawPane1. (Part 2 of 2.)

Deitel & Deitel (C) 2010 Pearson Education, Inc. All rights reserved.

### Java's Coordinate System

- class Graphics من الحزمة (java.awt)، والتي توفر طرقاً مختلفة لرسم النص والأشكال على الشاشة.
- الصنف JPanel من الحزمة (javax.swing)، والتي توفر مساحة يمكن الرسم عليها.

```
public class DrawPanel extends JPanel
```

`extends` الكلمة الأساسية للإشارة إلى أن الصنف DrawPanel هو نوع محسن من JPanel وارث له.

• الكلمة الأساسية `extends` تمثل ما يسمى بعلاقة الوراثة التي يبدأ فيها صنفنا الجديد DrawPanel بالأعضاء الحاليين (البيانات والأساليب) من فئة JPanel.

• كل لوحة JPanel بما في ذلك DrawPanel، لديها طريقة `paintComponent`.

• ينادي النظام تلقائياً في كل مرة يحتاج فيها إلى عرض DrawPanel المنهج `paintComponent()` ويجب التصريح عنها `public void paintComponent(Graphics g)`، خلاف ذلك، لن يسمح النظام بمناداتها والعبارة الأولى فيها عندما تكتبها (تحملها تحملاً زائداً) هي `super.paintComponent(g)`;

• يتم استدعاء هذه الطريقة عندما يتم عرض JPanel لأول مرة على الشاشة، وعندما يتم تغطيتها ثم الكشف عنها بواسطة نافذة أخرى على الشاشة، وعندما يتم تغيير حجم النافذة التي تظهر فيها.

# GUI and Graphics

## Creating Simple Drawings

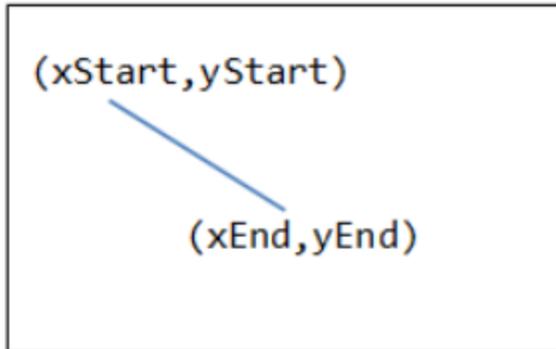
- تقوم أساليب `getWidth` و `getHeight` من الصنف `JPanel` بإرجاع عرض وارتفاع `JPanel` على التوالي.
- طريقة الرسم `drawLine` تحتاج أربع وسطاء، أول اثنين هما إحداثيات `x` و `y` تعبر عن نقطة البداية ونهاية واحدة، وآخر وسيطتين هما إحداثيات نقطة النهاية الأخرى وترسم خطاً بين نقطتي البداية والنهاية.
- لعرض لوحة الرسم `DrawPanel` على الشاشة، توضع في نافذة `place it in a window`.
- يتم إنشاء نافذة مع كائن من فئة `JFrame`.
- أسلوب `JFrame.setDefaultCloseOperation` مع الوسيطة `JFrame.EXIT_ON_CLOSE` يشير إلى أنه يجب إنهاء التطبيق عندما يغلق المستخدم النافذة.
- تقوم `add` method من `JFrame` بإرفاق `DrawPanel` أو (أي مكون GUI آخر) بإطار `JFrame`.
- تأخذ `setSize` method من `JFrame` معلمتين تمثلان عرض وارتفاع `JFrame`، على التوالي.
- تعرض `setVisible` method من `JFrame` مع الوسيط `true` الإطار `JFrame`.
- عندما يتم عرض `JFrame`، يتم استدعاء طريقة `paintComponent` الخاصة بـ `DrawPanel` ضمناً.

### Drawing Rectangles, Ovals

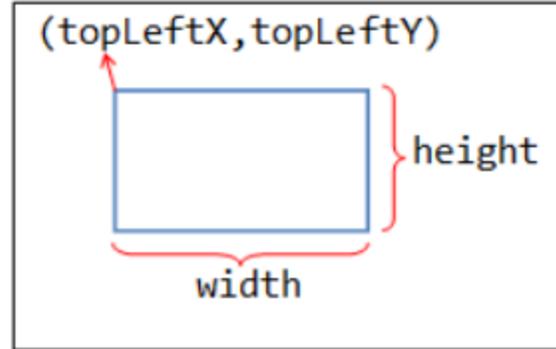
- يتطلب المنهج `drawRect` أربع وسطاء. يمثل أول اثنان إحداثيات  $x$  و  $y$  للزاوية اليسارية العلوية للمستطيل ، ويمثل الخياران التاليان عرض المستطيل وارتفاعه.
- البرنامج يرسم القطع الناقص. يقوم بإنشاء مستطيل وهي يسمى المستطيل المحيط ويضع بداخله شكل بيضاوي يلامس نقاط المنتصف لجميع الجوانب الأربعة.
- يتطلب أسلوب `drawOval` نفس الوسطاء الأربعة مثل طريقة `drawRect`.
- مثلاً لرسم مستطيل بحواف دائرية و مثلة ممتلئ باللون الاقتراضي نكتب:

```
g.drawRoundRect(200,150,60,50, 15,15);  
g.fillRoundRect(290,150,60,50,30,40);
```

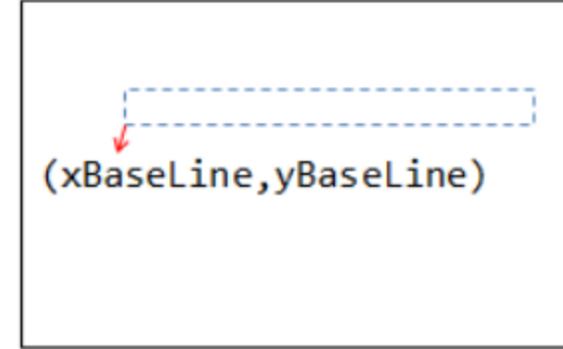
- في `OptionPane`، يجب عليك استخدام `\n` لبدء سطر جديد من النص ، بدلاً من `%n` ،
- يستخدم منهج `parseInt` لتحويل السلسلة التي أدخلها المستخدم إلى عدد صحيح ويخزن النتيجة في متغير للنوع الصحيح يتم اختياره.



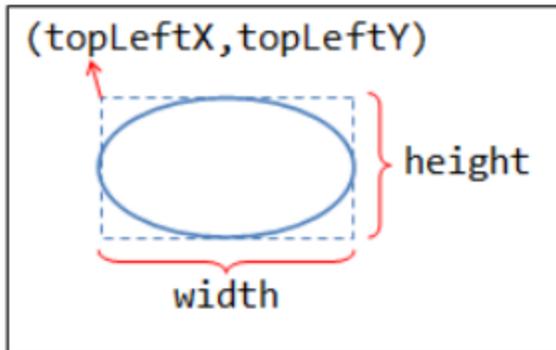
**drawLine()**



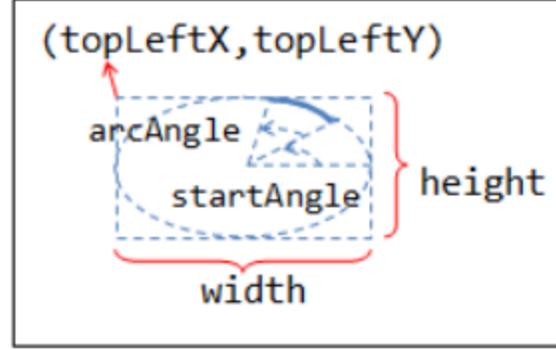
**drawRect()**



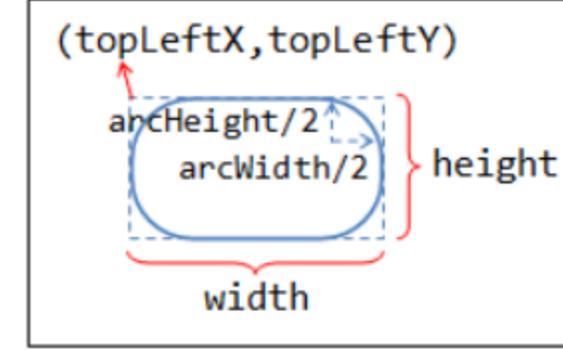
**drawString()**



**drawOval()**



**drawArc()**



**drawRoundRect()**



# Drawing Rectangles and Ovals



## 5.26 GUI & Graphics

```
1 // Fig. 5.26: Shapes.java
2 // Demonstrates drawing different shapes.
3 import java.awt.Graphics;
4 import javax.swing.JPanel;
5
6 public class Shapes extends JPanel
7 {
8     private int choice; // user's choice of which shape to draw
9
10    // constructor sets the user's choice
11    public Shapes( int userChoice )
12    {
13        choice = userChoice;
14    } // end Shapes constructor
15
```

**Fig. 5.26** | Drawing a cascade of shapes based on the user's choice. (Part I of 2.)

```

16 // draws a cascade of shapes starting from the top-left corner
17 public void paintComponent( Graphics g )
18 {
19     super.paintComponent( g );
20
21     for ( int i = 0; i < 10; i++ )
22     {
23         // pick the shape based on the user's choice
24         switch ( choice )
25         {
26             case 1: // draw rectangles
27                 g.drawRect( 10 + i * 10, 10 + i * 10,
28                             50 + i * 10, 50 + i * 10 );
29                 break;
30             case 2: // draw ovals
31                 g.drawOval( 10 + i * 10, 10 + i * 10,
32                             50 + i * 10, 50 + i * 10 );
33                 break;
34         } // end switch
35     } // end for
36 } // end method paintComponent
37 } // end class Shapes

```

Draws a rectangle starting at the x-y coordinates specified as the first two arguments with the width and height specified by the last two arguments

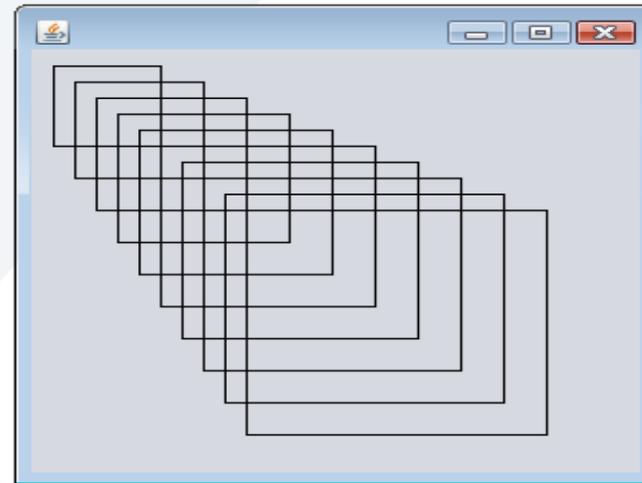
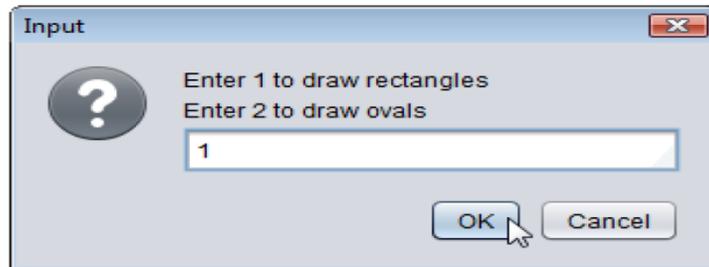
Draws an oval in the bounding rectangle starting at the x-y coordinates specified as the first two arguments with the width and height specified by the last two arguments

**Fig. 5.26** | Drawing a cascade of shapes based on the user's choice. (Part 2 of 2.)

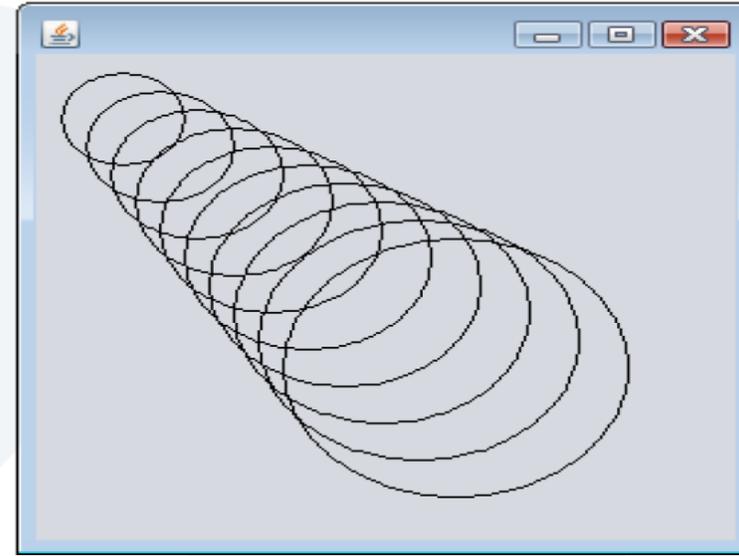
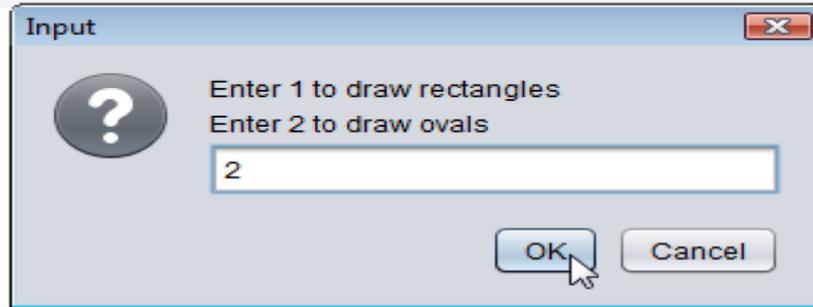
```
1 // Fig. 5.27: ShapesTest.java
2 // Test application that displays class Shapes.
3 import javax.swing.JFrame;
4 import javax.swing.JOptionPane;
5
6 public class ShapesTest
7 {
8     public static void main( String[] args )
9     {
10         // obtain user's choice
11         String input = JOptionPane.showInputDialog(
12             "Enter 1 to draw rectangles\n" +
13             "Enter 2 to draw ovals" );
14
15         int choice = Integer.parseInt( input ); // convert input to int
16
17         // create the panel with the user's input
18         Shapes panel = new Shapes( choice );
19
20         JFrame application = new JFrame(); // creates a new JFrame
21
```

**Fig. 5.27** | Obtaining user input and creating a JFrame to display Shapes. (Part 1 of 3.)

```
22     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
23     application.add( panel ); // add the panel to the frame
24     application.setSize( 300, 300 ); // set the desired size
25     application.setVisible( true ); // show the frame
26 } // end main
27 } // end class ShapesTest
```



**Fig. 5.27** | Obtaining user input and creating a JFrame to display Shapes. (Part 2 of 3.)



**Fig. 5.27** | Obtaining user input and creating a JFrame to display Shapes. (Part 3 of 3.)

## *Java's Coordinate System*

# GUI and Graphics

## Drawing Rectangles, Ovals

```
// Fig. 5.28: ShapesTest.java
// Obtaining user input and creating a JFrame to display Shapes.
import javax.swing.JFrame; //handle the display
import javax.swing.JOptionPane;
public class ShapesTest
{ public static void main(String[] args)
    {
        // obtain user's choice
String input = JOptionPane.showInputDialog(
"Enter 1 to draw rectangles " +
"Enter 2 to draw ovals");
int choice = Integer.parseInt(input); // convert input to int
```

## *Java's Coordinate System*



# GUI and Graphics

## Drawing Rectangles, Ovals

```
// create the panel with the user's input
Shapes panel = new Shapes(choice);

JFrame application = new JFrame(); // creates a new JFrame

application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
application.add(panel);
application.setSize(300, 300);
application.setVisible(true);
}
} // end class ShapesTest
```

```
/ Fig. 5.26 A: Shapes.java
// Demonstrates drawing different shapes.
import java.awt.Graphics;
import javax.swing.JPanel;
public class Shapes0 extends JPanel
{
    private int choice; // user's choice of which shape to draw
    // constructor sets the user's choice
    public Shapes0( int userChoice )
    {
        choice = userChoice;    } // end Shapes constructor

    // draws a cascade of shapes starting from the top left corner
    public void paintComponent( Graphics g )
    {
        super.paintComponent( g );

        for ( int i = 0; i <= 10; i++ )
        {
            // pick the shape based on the user's choice
```



```

switch ( choice )
{
  case 1: // draw rectangles
    g.drawRect( 10+i*10,10+i*10,50+i*10, 50 + i * 10 ); break;
  case 2: // draw ovals
    if(i%4==0)g.fillOval(10+i*10, 10+i*10, 50 + i * 10, 50 + i * 10 );
    else
      g.drawOval( 10+i*10, 10+i*10, 50+i*10, 50 + i * 10 ); break;
  case 3: //draw line fig4.20a Lines fanning from a corner. P138
    g.drawLine( 10, 10,260 - i * 25, 10 + i * 25); break;
  case 4: //draw line fig4.20b Lines fanning from a corner.P138 9th
    g.drawLine( 10 , 10, 260 - i * 25, 10 + i * 25);
    g.drawLine( 260, 260,260 - i * 25, 10 + i * 25);
    g.drawLine( 10, 260, 10 + i * 25, 10 + i * 25);
    g.drawLine( 260, 10, 10 + i * 25, 10 + i * 25); break;
  case 5: // draw Fig4.21a Line art with loops and drawLine. P138 9th
    g.drawLine( 10 , 10 + i*25, 10 + i * 25, 260); break;
}

```





```
case 6: //draw Fig4.21b Line art with loops and drawLine. P138 9th
g.drawLine( 10 , 10 + i*25,10 + i * 25, 260);
g.drawLine( 10 + i*25, 10, 260 , 10 + i * 25);
g.drawLine( 260 -i*25, 10, 10 , 10 + i * 25);
g.drawLine( 10+i*25, 260 , 260 ,260 - i*25); break;

case 7: // draw concentric circles Fig. 5.29 P190 10th
g.drawOval( 130 - i * 10, 130 - i * 10,i * 20, i * 20 );
// g.drawOval(30+i*10, 30+i*10, 200-i * 20, 200- i * 20 );
int x1,y1,w,h;
x1=130 - i * 10; y1=130 - i * 10;w= i * 20; h=i * 20;
System.out.println("x1="+x1+" y1= "+y1+" w= "+w+" h= "+h);break;

case 8: g.drawString("Welcom", 20, 20);
// Draw a right triangle of stars in the output window,
// do not use the Graphics class
for ( int a = 1; a <= 5; a++ ) {
for ( int j = 1; j <= a; j++ ) {
System.out.print( '*' ); } // end inner for
```





جامعة  
المنارة

```
System.out.println(); } // end outer for
    break;
case 9: g.drawString("Welcom", 10 + 10*i, 10 + 10*i);
case 10: // draw ovals
    if(i%4==0)g.fillOval(10 + i * 10, 10 + i * 10,
        50 + i * 10, 50 + i * 10 );
    else g.drawOval( 10+i*10, 10+i*10, 50+i*10, 50+i*10); break;
case 11: // draw rectangles
    if(i<4)g.drawRect( 10+i*10, 10+i*10, 50 + i * 10, 50 + i * 10 );
    if(i>=4 && i<7) g.fillRect( 10 + i * 10, 10 + i * 10,
        50 + i * 10, 50 + i * 10 );
    if(i>7)g.drawRoundRect(10+i*10,10+i*10, 50+i*10, 50+i*10,20,30);
        break;
    } // end switch
} // end for
} // end method paintComponent
// end class Shapes
```



```
// Fig. 5.27 A : ShapesTest0.java
// Test application that displays class Shapes.

import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class ShapesTest0
{
    public static void main( String args[] )
    {
        // obtain user's choice
        String input = JOptionPane.showInputDialog(
            "Enter 1 to draw rectangles Fig 528a\n" + "Enter 2 to draw ovals Fig 528b \n"
            + "Enter 3 to draw Fig 420a 10 line \n" + "Enter 4 to draw Fig 420b 40line \n"
            + "Enter 5 to draw Fig 421a P139 \n"+"Enter 6 to draw Fig 421b P139 \n"
            + "Enter 7 to draw Fig. 5.29 P190 \n"+"Enter 8 to draw for E5.15 p196a \n"
            + "Enter 9 to drawString \n" + "Enter 10 to drawOval and 2 fillOval \n"
            + "Enter 11 to drawRect and fillRect and drawRoundRect \n");
    }
}
```

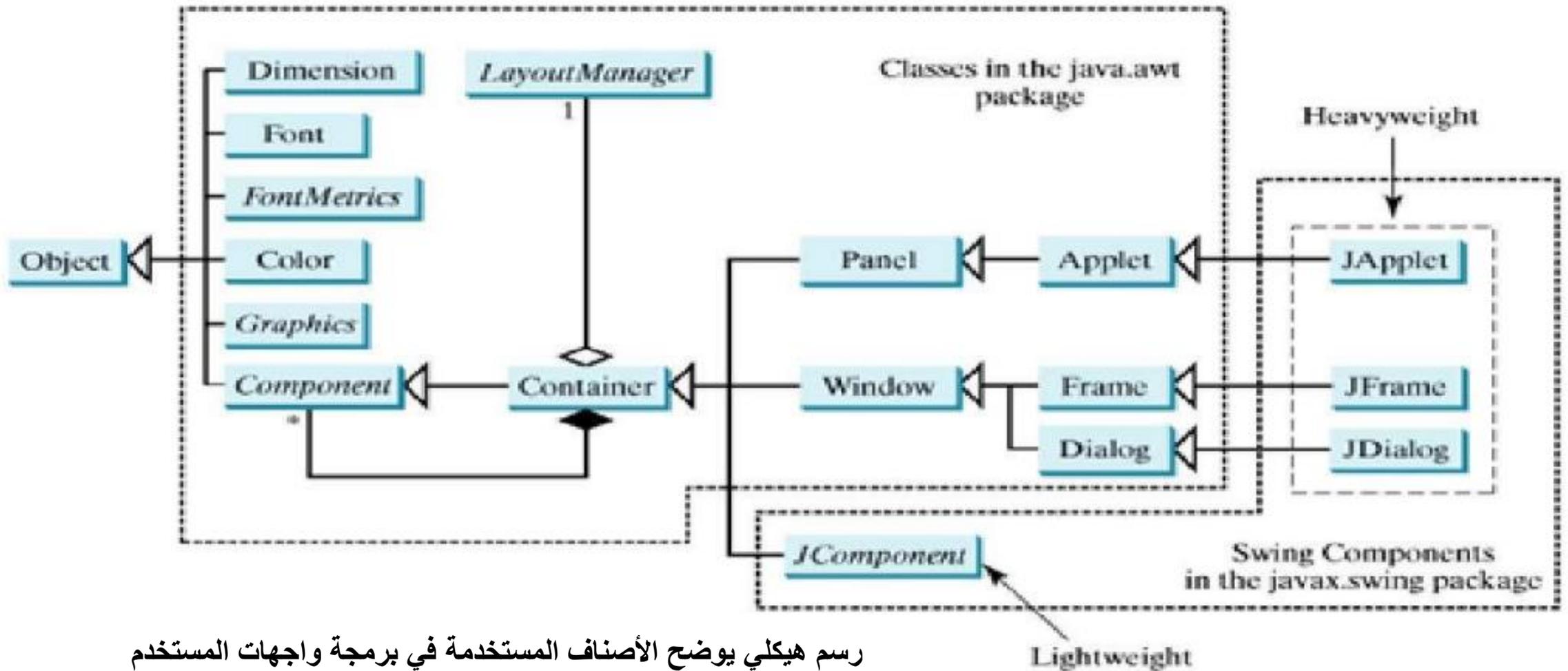
```
int choice = Integer.parseInt( input ); // convert input to int

// create the panel with the user's input
Shapes0 panel = new Shapes0( choice );

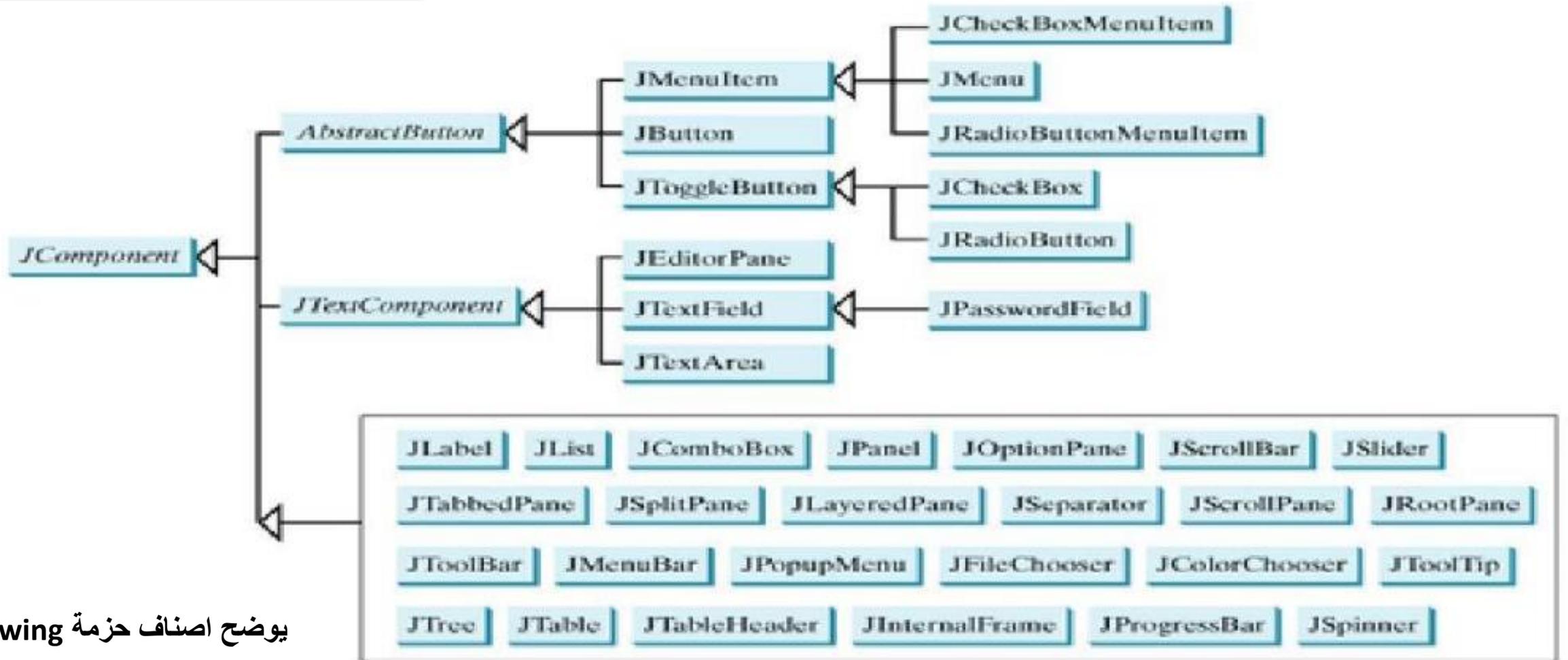
JFrame application = new JFrame(); // creates a new JFrame

application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
application.add( panel ); // add the panel to the frame
application.setSize( 280, 305 ); // set the desired size
application.setVisible( true ); // show the frame
} // end main
} // end class ShapesTest
```

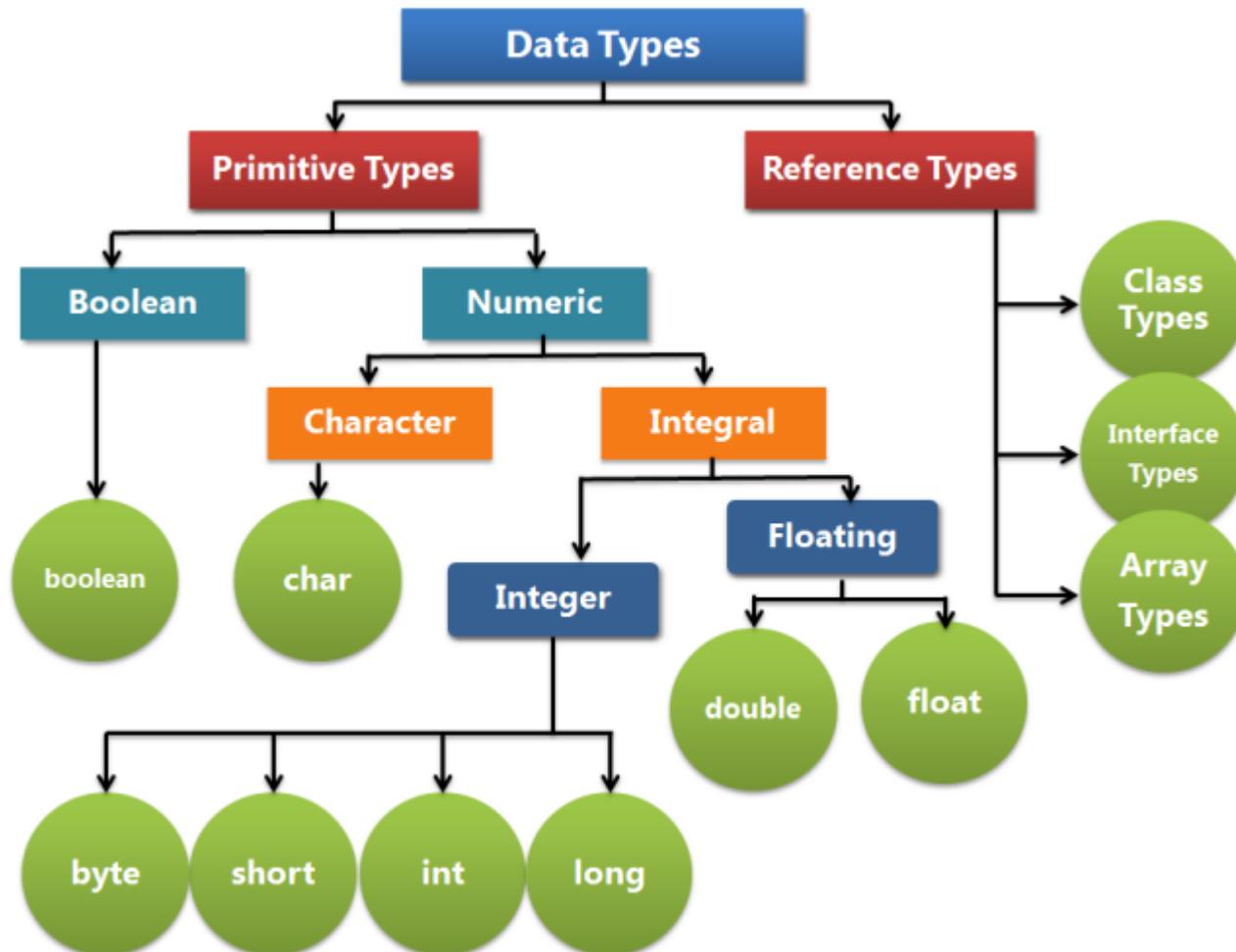
# انتهت محاضرة الأسبوع 3

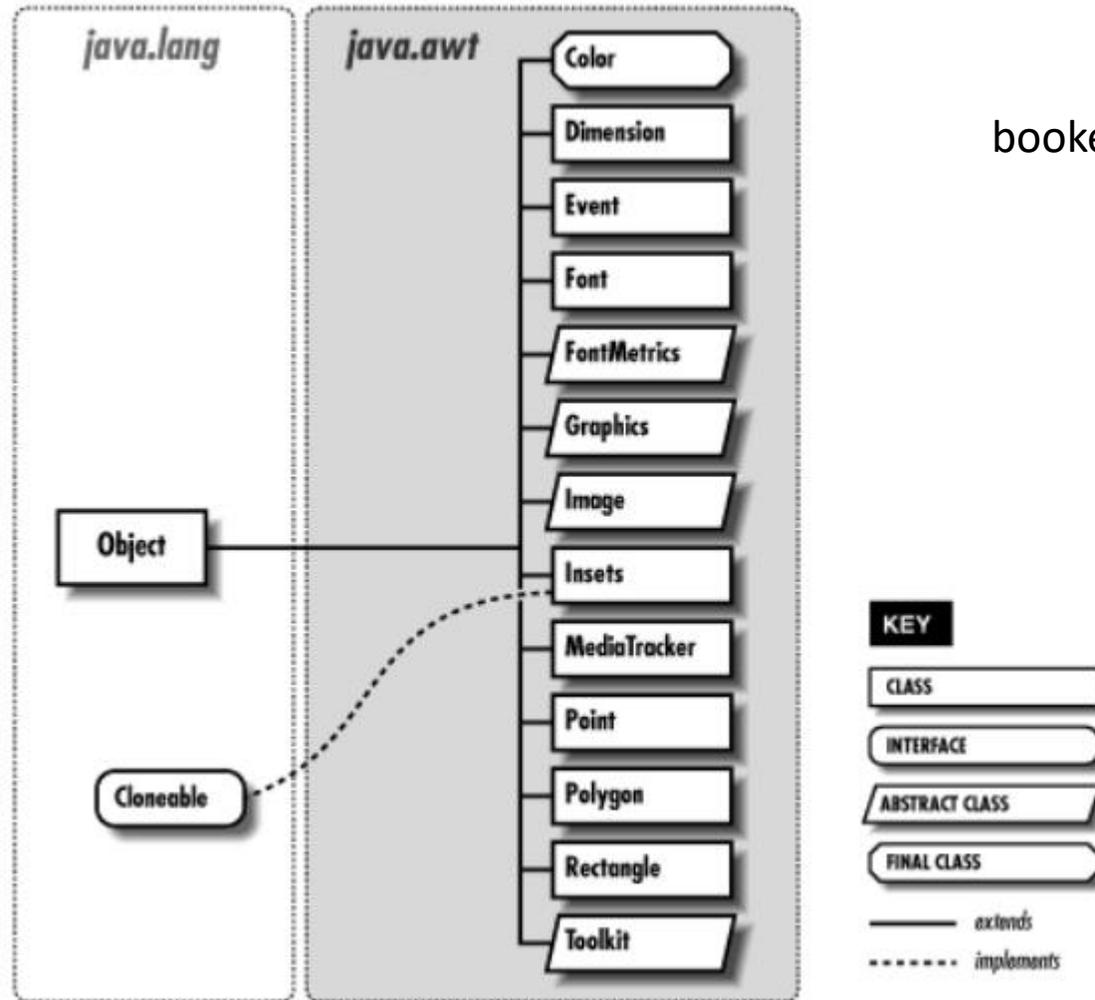


رسم هيكل يوضح الأصناف المستخدمة في برمجة واجهات المستخدم



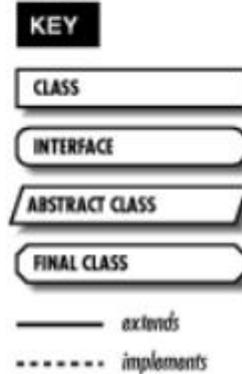
يوضح اصناف حزمة swing

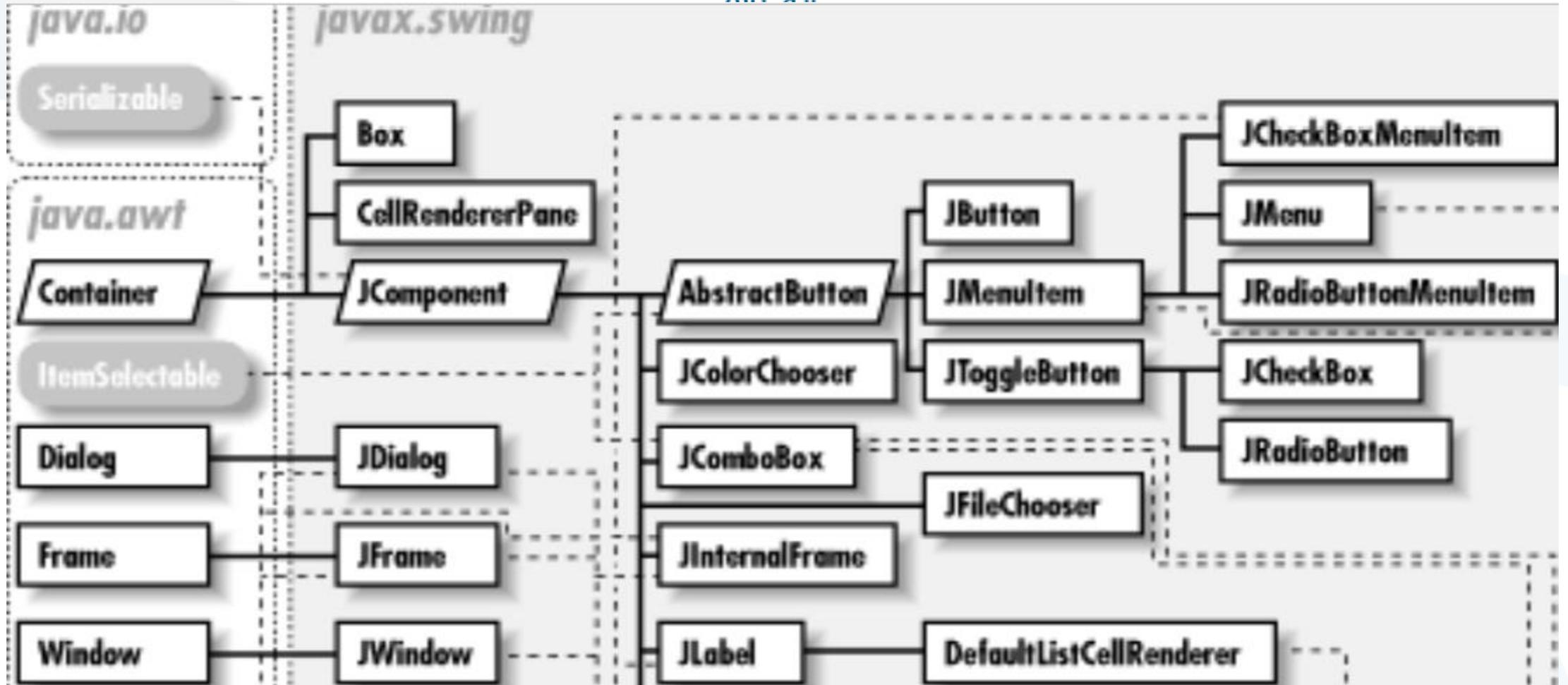


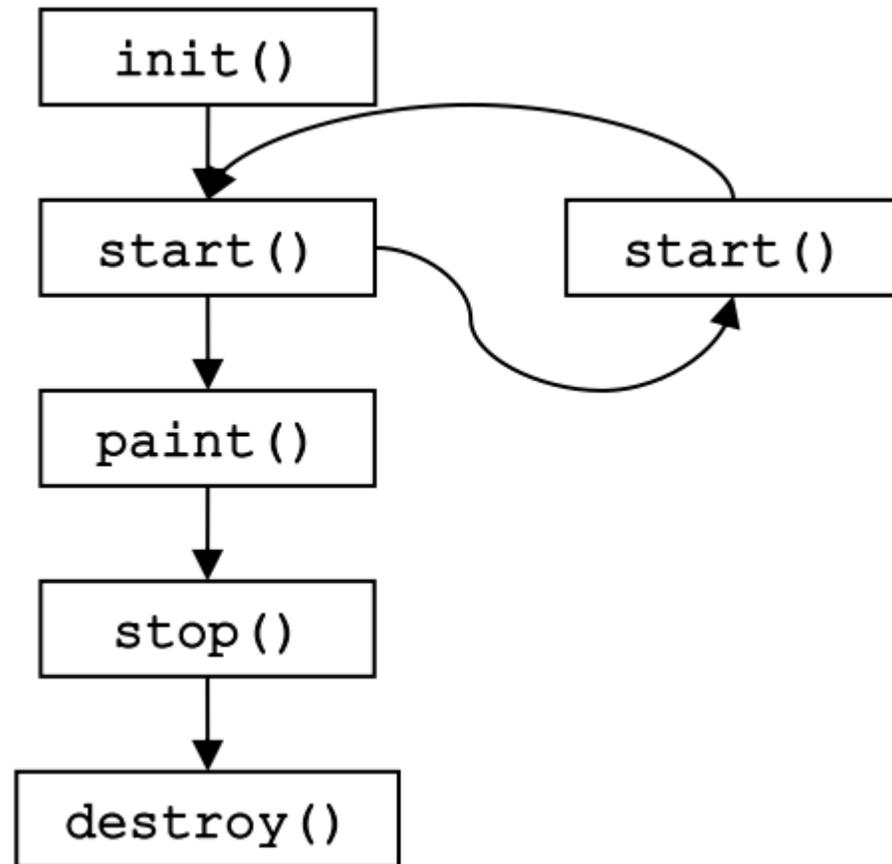


bookes\dataStru\java2021-2022\scrap

Noor-Book.com  
 مدخل أساسي إلى البرمجة  
 كائنية التوجه أساسيات  
 البرمجة بلغة جافا  
 Pag 96 Java







## Chapter 6.13 p227

```
public Color(int r, int g, int b)
```

Graphics methods fillRect and fillOval draw filled rectangles and ovals.

