جَامعة المَنارة
MANARA UNIVERSITY

كلية الهندسة المعلوماتية

برمجة 3

Java Programming

ا. د. علي عمران سليمان

المحاضره 2 من محاضرات الأسبوع الثاني

Polymorphism & Interfaces

الفصل الصيفي 2024-2025

content

- Polymorphism
- 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، بنى معطيات بلغة JAVA، جامعة تشرين 2013-2014

Method Overriding
(replacement or expansion)

- يوجد نوعين Method Overriding

1-هما الاستيدال replacement 2- التحسين refinement(التوسيع expansion)

- الاستيدال 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 لإن كل طريقة جديدة هي استبدال للطريقة الموروثة.

- التحسين أي تضيف شيفرة إضافية إليها: يستخدام constructors طريقة التحسين في إعادة تعريف الطرائق ويدعى قطر توابع البناء constructors chaining واستدعاء الباني الافتراضي يتم تلقائياً، (مثال أخر: استخدام طريقة ()printAllDatails لطباعة معلومات Person وتوسيعها ضمن الصنف Employee لإكمال مايخص الموظف وأول سطر هو مناداة طريقة طباعة Person وفق التالي ;()super. printAllDatails ثم كتابة مايخص 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 موظف Employee ويشير لكائن من نوع موظف.

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 طريقه سيتم تنفيذها .

- إن ()get_salary موجوده ضمن الصنف الأب وبالتالي معرفة على كل الأصناف الوارثة له.
- بفرض أن طريقة ما معرفة ضمن SalariedEmployee وتم ندائها 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.
- لايمكن لمرجع من الصنف المشتق SalariedEmployee أن يؤشر إلى كائن من الصنف الأساس بدون تحويل تضييق DowenCasting

SalariedEmployee e3= new Employee (“adam”,30,”Hama”, …); // Error

يوجد نوعين من التحويلات:

- تحويل التوسيع widening أو UpCasting او Implicit Casting ويحصل بشكل تلقائي أو automatic عندما يشير مؤشر من نمط الاب إلى كائن من نمط الابن، وبشكل طبيعي كل SalariedEmployee هو Employee والعكس ليس صحيحاً،

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

وكمثال للحالة السابقة التي لاتحتاج للاختبار كون SalariedEmployee هو وارث Employee:

```
Employee e1= new Employee (“adam”,30,”Hama”, …);
SalariedEmployee e3= (SalariedEmployee ) e1;
```

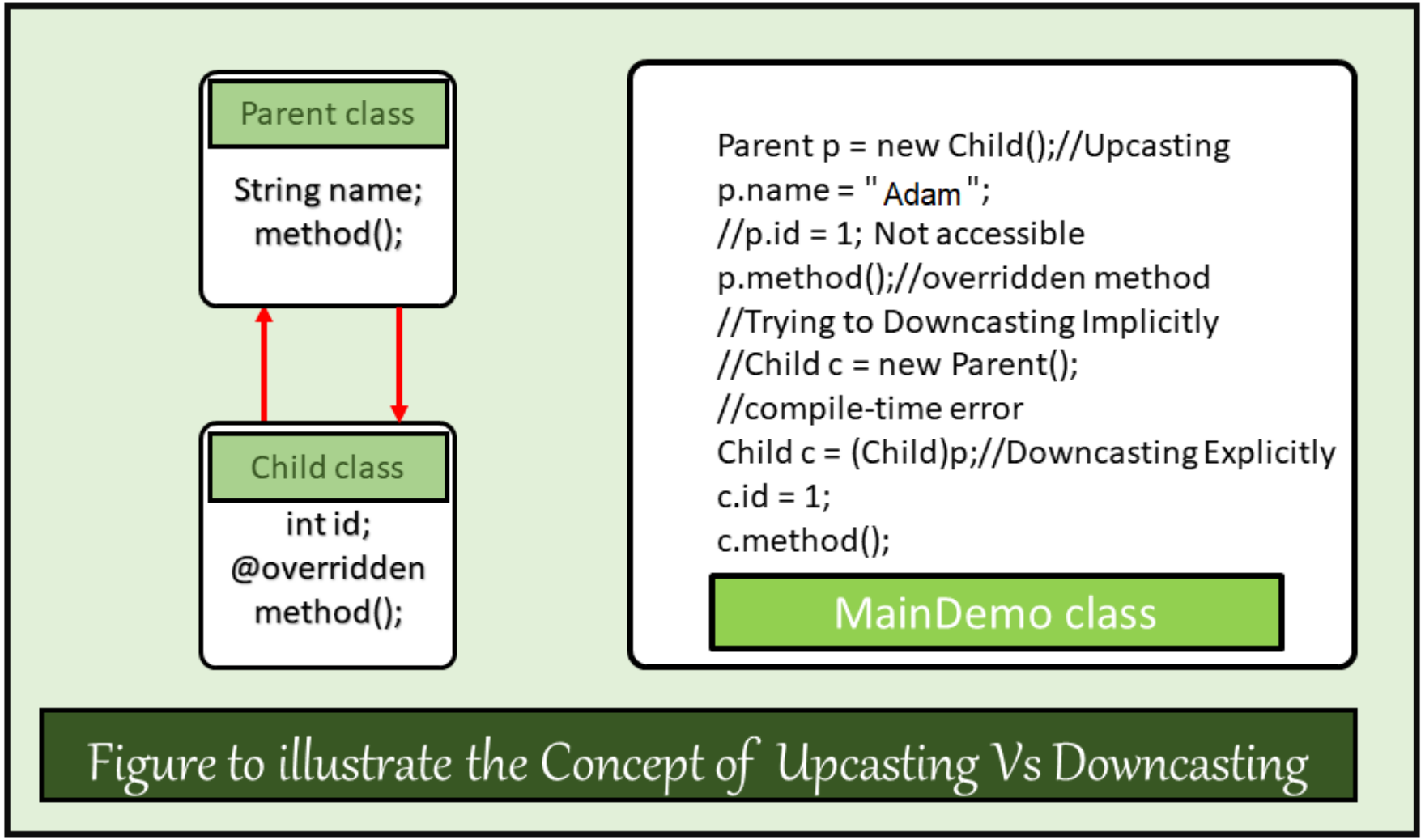


Figure to illustrate the Concept of Upcasting Vs Downcasting

Polymorphism

يوجد لدينا نوعين من تعدد الأشكال :

1- static polymorphism هو Overloading. ويتم تحديد المنهج من خلال compiler .

2- (dynamic or binding) polymorphism هو Overriding ويتم تحديد المنهج من خلال runtime.

لنفرض لدينا البنية التالية: الصنف Object يرثة الصنف Person والصنف Person يرثة الصنفStudent وموروث منه GraduateStudent. الصنف Object هو بمثابة superClass ولدينا ()methods toString التي تعيد String

ولدينا ()toString موجود في الصنف الأعلى ومعمول لها Overriding في الصنفين Student & Person .

وفي الصنف GraduateStudent لم يتم اجراء Overriding لها. وبالتالي سينفذ أقرب نسخة له أي عند الأب المباشر له إن وجده وهو في Student في حالتنا وإلا سيبحث عند الجد وهكذا.

ولدينا كائن من الصنف Object اسمه x وهو superType للجميع وتم نداء المنهج اربع مرات وارسال لها كائنات من نمط مختلف ومن الأسفل باتجاه الأعلى وهنا تمت كتابة *polyPrint(*Object x*)* مره واحدة فقط وتنفيذها عدة مرات وفق نوع الكائن x المرسل لها ولم نحتاج لتكرار كتابتها لكل نمط من الأنماط الابع وهذا مايعرف Generic Programming. وسيتم تحديد النسخة المناسبة من خلال JVM وهو مايعرف dynamic binding .

Polymorphism

```
public class Person extends Object {public String toString(){return "Person";}}
public class Student extends Person {public String toString(){return "Student";}}
public class GraduateStudent extends Student{ }
public class PolymorphismDemo {        public static void main(String[] args) {
      polyPrint(new GraduateStudent()) ;     polyPrint(new Student ()) ;
      polyPrint(new Person());               polyPrint(new Object()); }
public static void polyPrint(Object x) {     System.out.println(x.toString());} }
```

Output:

Student
Student
Person
java.lang.Object@c17164

Abstract Classes 1

- 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 للاصناف الأخرى، كمثال لنفرض اننا نملك SalariedEmployeeأو 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 الفرعيه غيرالمجردة، وغير ذلك سنحصل على خطأ مطابقة.
- أي صنف يحتوي على طريقة مجردة abstract يصبح تلقائيًا 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.

Abstract Classes 2

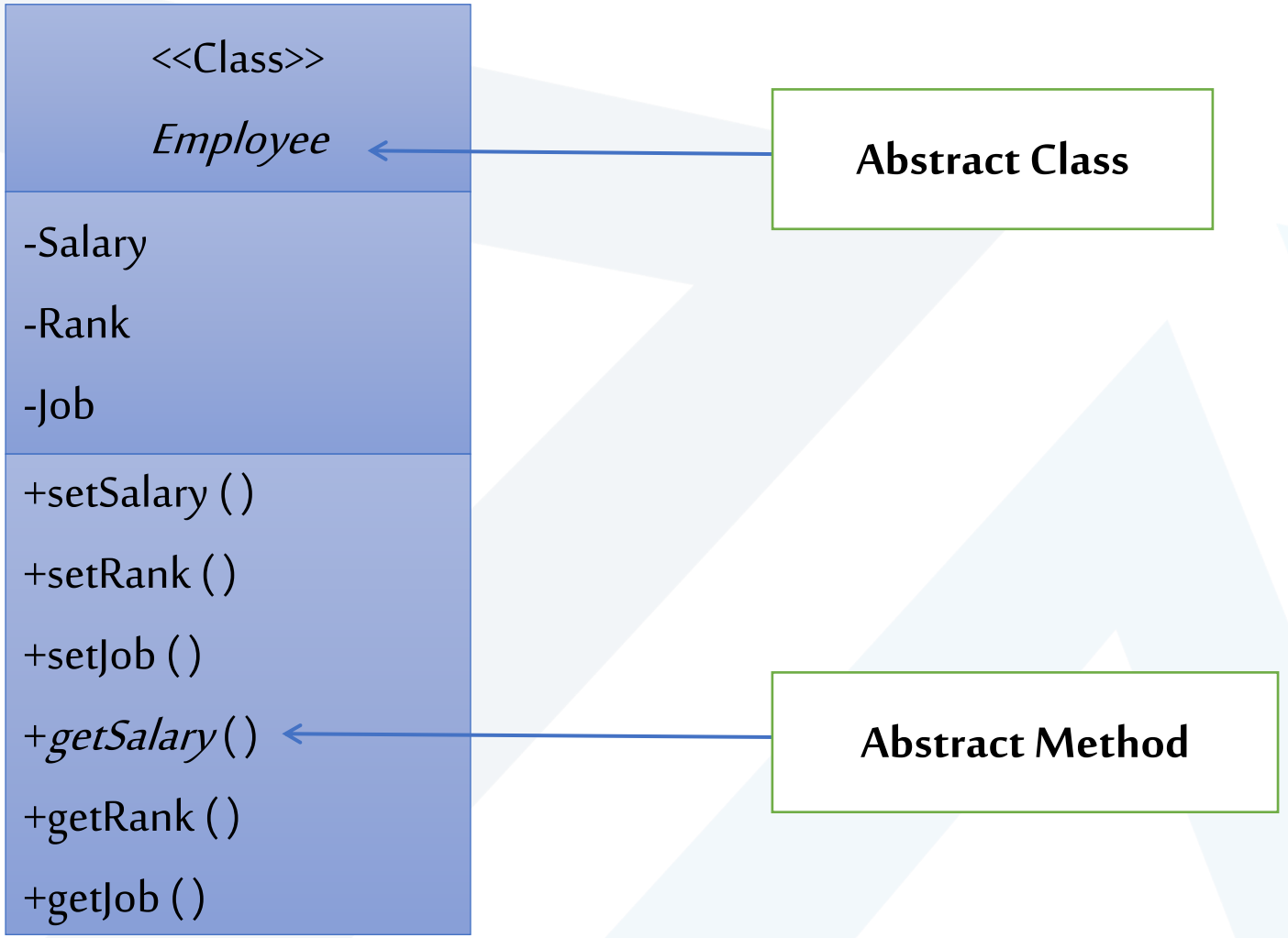
- يمثل الصنف المجرد الشكل العام أو الشكل المجرد لجميع الأصناف المشتقة منه.
- الشكل العام للطريقة المجرده:

AccessSpecifier abstract ReturenType MethodName(ParameterList); // without a body {}

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 تكرر 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 خلال الترجمة.

Abstract Methods 3



an exercise

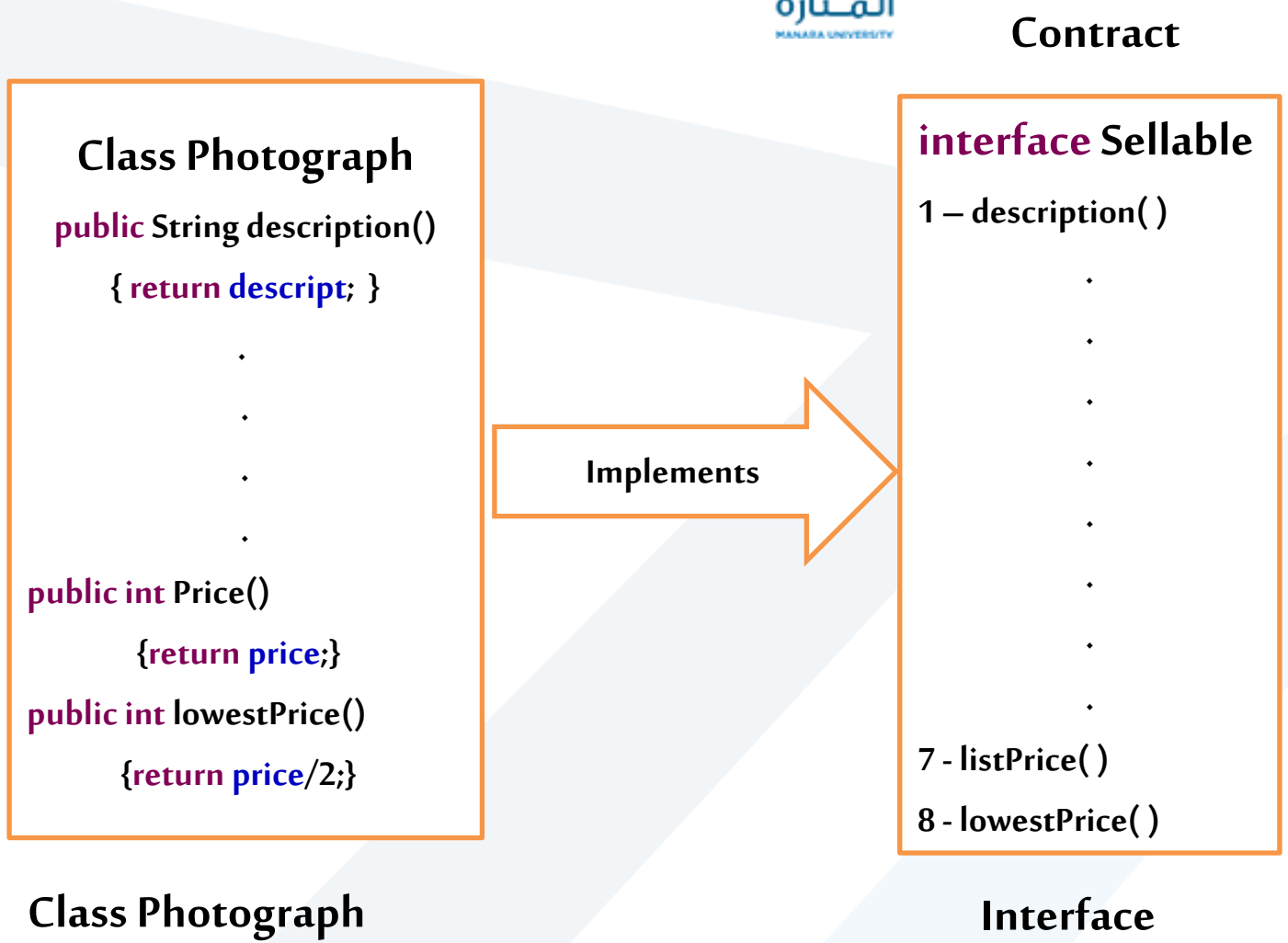
- لتحويل الطريقة getSalary() في الصنف Employee إلى طريقة مجرده توضع abstract ما قبل القيمة المعادة وحذف جسمها،
- سيحصل خطأ يطالب بتحوبل الصنف Employee المدروس سابقاً لصنف مجرد ويتم بوضع abstract مابعد نوع الوصول والكلمة المفتاحية class ستلاحظ عدم إمانية اشتقاق كائن منه.
- عند اشتقاق صنف غير مجرد SalariedEmployee من الصنف المجرد Employee ستجد وجود خطأ يطالب بإعادة كتابة الطريقة getSalary() لإنها طريقة مجرده.
- هذا ينطبق على الصنف HourlyEmployee ايضاً، وكل الآصناف المشتقة من الصنف المجرد Employee.
- إذا احتوى الصنف على عدة طرق مجردة يستحسن تحويل الصنف إلى واجهه Interfaces وهذا ماسنهتم به.
- الشكل العام لتعريف الواجهة:

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

Interfaces 1

- الواجهة interface تعميم للاصناف المجردة وتحتوي طرق مجردة.
- الغاية من الواجهة interface هو تحديد السلوكيات للاصناف الأخرى المحققه لها.
- الواجهة interface. هي عبارة عن مجموعة من تصريحات الطرائق بدون أي أجسام لها، أي أن طرائق الواجهة يكتب نموذجها فقط proto type (أي تواقيع الطرائق فقط).
- 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، بهذا الاسلوب، فإن الواجهات تتطلب وجود صنف بناء يملك طرائقها بنفس التواقيع المحددة.
- يقال غالبًا أن الواجهة تشبه "العقد" contract، وعندما ينفذ صنف لواجهة ، يجب أن يلتزم بالعقد.

Interfaces 2



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

Interfaces 3

- 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 كتابةالصنف 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 والسبب إن طرائق الواجهات ليس لها أجسام تعريف وبالتالي لن يحصل التباس عند وجود طريقتين بنفس التوقيع في واجهتين مختلفتين نظراً لعدم وجود اجساد لها في الواجهات المورثة، ويتحقق ذلك بصنف يحقق عدة واجهات أو واجهه ترث عدة واجهات ومن ثم تحقيقها.

Interfaces 3

- تحتوي جميع الكائنات في Java على طريقة خاصة تسمى toString والتي تعرض تمثيل الشريط المحرفي String لمحتويات الكائن.
- الطريقة toString موجوده ضمن الصنف Object وتسمى هذه الطرق بطرق الخدمات العامة أيضًا، أو الواجهة العامة التي يوفرها الصنف لعملائه.
- عندما يتم ربط كائن بسلسلة عن طريقة overridden، يتم استدعاء toString ضمنيًا 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 المعبرعن كل صنف كمثال :

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

وفي حال عدم كتابتها ستعطي مثلاً Pho1 @1fee6fc اسم الكائن وعنوانه ضمن الذاكرة والعائدة للصنف المورث Object.

Multiple Inheritance in Interfaces 2

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 والمحقق لها ومن أجل التحف المتنوعة التي يمكن أن نبيعها، (نحزمها ونشحنها). وبالتالي، يبني الصنف BoxedItem طرائق الواجهة Sellable والواجهة Transportable مع إضافة طرائق مخصصة لتحديد قيمة التأمين لشحن صندوق و أبعاد 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=1;             // 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 );
               Sellable boxi1= new BoxedItem("adamm", 1090,100, true );
               System.out.println(boxi1);
               box1= (BoxedItem)boxi1;     // Down Casting   box1 referents of class BoxedItem  , boxi1 referents  interphase  Sellable
               System.out.println("/n/n");System.out.println(box1);
                box1.setBox(3, 5, 7); System.out.println("\n\n");
               System.out.println(box1.toString());
       } // end maim
}// end class BoxedItem
```

Multiple Inheritance in Interfaces 1

- عندما ينفذ صنف واجهات متعددة، يجب أن توفير الطرق المحددة من قبل كل منهم.
- لتحديد واجهات متعددة في تعريف صنف، تسرد أسماء الواجهات ، مفصولة عن بعضها بعضاً بفواصل، بعد الكلمة المفتاحية implements كما في المثال السابق.
- بالعودة إلى مثال التحف، يمكن أن نعرف واجهة للأغراض المؤمن عليها ترث الواجهتين 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 لنا أن نعرف الصنف BoxedItem2 لتحقيقها:

```
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، لإنها معرفه ضمن الواجهه الوريثة 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.

• عند تحويل التضيق Narrowing أو DowenCasting أو ExplicitCasting يجب التأكد من إمكانية القسر قبل اجراء القسر .

يوجد عامل مقارنة النوع يختبر ذلك instanceof ويعيد قيمة منطقية فإذا كانت true نقوم بالقسر الآمن

(object) instanceof (type)

تعرف بـ (RTTI) run – time type identification

Multiple Inheritance in Interfaces 3

```
//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  child class
@Override
void method()
{System.out.println("Method from Child");}  }
// Demo class to see the difference between upcasting and downcasting

public class GFG {public static void main(String[] args)
{ Parent p = new Child(); p.name = "GeeksforGeeks";          /* Upcasting */
System.out.println(p.name);                          /*Printing the parent class 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();}
} // end class parent
```

• بفرض أننا صرحنا عن الواجهة Personالمبينة 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 المعلنة في الواجهة تكون متاحة،بما يحاكي المعروض في الشريحة الرابعة.

• 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 تضييق من النمط Person (الواجهة) إلى النمط Student (الصنف) يتطلب 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 + ", studyHours " +
(double)age/2 + ")";
            }          // 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 تضييق من أنماط واجهات إلى أنماط اصناف تتيح لنا كتابة أنواع عامة من بنى 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 بمايلي (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 تضييق 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 Personالمعادة 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 بنى معطيات عامة، والتي يمكن تخصيصها من قبل مبرمجين آخرين من خلال استخدام تحويل الأنماط.

Polymorphism with Interfaces

- 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 pho2, pho1 .
- يشير المتغير المرجعي pho1 إلى كائن Photograph ويشير متغير pho2 إلى كائن BoxedItem.
- عندما ينفذ صنف واجهة ، يتم إنشاء علاقة وراثة تعرف باسم وراثة الواجهة.

```
Sellable pho1= new Photograph ();
Sellable pho2= new BoxedItem ();
```

- عند ذكر .pho1 ستظهر كل الطرق المتاحة والمعرفة ضمن الواجهه المحققة.
- الطرق المعرفة ضمن الصنف المحقق للواجهه لن يتم التعرف عليها لعدم معرفة الواجهه بها مثل pho1.isColor(); على غرار مرجع من نوع ويشير على كائن من نوع مشتق لن يصل للطرق التي لم تعرف في الأصل الشريحة أربعة.

Fields in Interfaces

- 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.

ملاحظة: عند معرفة القيمة الثابتة قبل المطابقة يستخدم الواصف const وعندما لايعرف حتى وقت التنفيذ يستخدم final .

التعميم Generics

- في إطار عمل التعميمات generics framework لاستخدام الأنماط المجردة بطريقة تتجنب العديد من التحويلات الصريحة.
- النوع العمومي generic type هو نوع لايعرف في زمن الترجمة، و إنما 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، مثلاً: *لتجريد أنماط بعض المتحولات الداخلية للصنف، تستخدم أقواس زاوية angle brackets لحصر قائمة بالبارامترات ذات الأنماط الشكلية*، على الرغم من أن أي معرف مرجعي صالح يمكن أن يستخدم من أجل بارامتر ذو نمط شكلي.
- الصيغة العامة لإنشاء كائنات من صنف عام:

BaseType <Type> obj = new BaseType <Type>() // To create an instance (obj)of generic class BaseType

Note: <Type> is classType as (Integer, Character, Class defined by User, ...) not primitive type as (int, double, char, ...) ... etc.

Test<int> obj = new Test<int>(20); // compile-time error, that can be resolved using type wrappers to encapsulate a primitive type.

ArrayList<int[]> a = new ArrayList<>(); //primitive type arrays can be passed to the type parameter **because arrays are reference types.**

- يبين المقطع البرمجي التالي الصنف Pair الذي يخزن أزواجاً مفتاح–قيمة pairs(key, value) حيث إن أنواع القيمة والمفتاح محددة بالبارامترات V وK على التوالي. تقوم الطريقة main بإنشاء مثلين من هذا الصنف، الأول من أجل زوج String-Integer والثاني Student-Double .

Generic Class

```
public class Pair<K, V> { K key;  V value;                    // Generic class
public void set(K k, V v){key= k; value= v;}                  // Generic method
public K getKey() { return key; }                             // Generic method
public V getValue() { return value; }          // The common type parameters are : T:Type,  E: Element, K: Key, N: Number, V: 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(29.5));  System.out.println(pair2);
        //System.out.println("\n\n"+pair2.equals(pair1));
        Pair<Student , Double> pair3=new Pair<Student , Double>();
         pair3.set(new Student("A5976","Sue",19),new Double(29.5));
         System.out.println("\n"+)pair2.key.equalTo(pair3.key);
              }// end main
}             // class Pair
```

```
[height, 36]
[Student(ID: A5976, Name: Sue, Age: 19, studyHours 9.5), 29.5]

true
```

التعميم Generics

مزايا Generics Type:

- **إمكانية إعادة استخدام الكود:** يمكننا كتابة دالة أو فئة أو واجهة مرة واحدة واستخدامها مع أي نوع.
- **سلامة النوع:** تضمن الصيغ العامة اكتشاف الأخطاء أثناء التجميع compiler بدلاً من التشغيل RTE، مما يعزز أمان الكود.
- **لا حاجة لتحويل النوع:** يعالج المترجم التحويل تلقائيًا، مما يلغي الحاجة إلى تحويل النوع الصريح عند استرداد البيانات.
- **سهولة قراءة الكود وصيانته:** من خلال تحديد الأنواع وجعل الأنواع المقصودة من هياكل البيانات والأساليب واضحة، يصبح الكود أسهل في القراءة والصيانة.
- **الخوارزميات العامة:** تتيح الصيغ العامة تنفيذ خوارزميات تعمل عبر أنواع مختلفة، مما يعزز كفاءة ممارسات البرمجة.

عيوب Generics Type:

- **التعقيد:** يمكن أن تُضيف الأنواع العامة تعقيدًا إلى الكود، مثل فهم مفاهيم الأحرف البديلة (? extends, ? super)صعبًا على المبتدئين.
- **تكلفة الأداء:** يُسبب مسح النوع قد تفرض بعض التكلفة الإضافية نظرًا لتحويل الأنواع العامة إلى كائنات أثناء التشغيل.
- **عدم دعم الأنواع الأولية:** تعمل الأنواع العامة فقط مع أنواع المراجع، مما يتطلب استخدام فئات التغليف مثل Integerأو Double للأنواع الأولية وبالتالي قد يفرض تكلفة اضافية.
- **انعكاس محدود:** يحد مسح النوع من مدى إمكانية استخدام الانعكاس مع الأنواع العامة، نظرًا لعدم توفر معلومات النوع أثناء التشغيل.
- **القيود:** بعض القيود على كيفية استخدامها، مثل عدم القدرة على إنشاء مصفوفات من الأنواع ذات المعلمات أو declare static fields of type parameters

انتهت محاضرات الأسبوع الرابع

لتحويل الأنماط الأولية نتبع

```
String str =“200”; int num=Integer.Int(str);
System.out.println(num+num); // 400
int num=100; String strNum=String.valueOf(num);
System.out.println(strNum+9);  // 1009
int num=100;  System.out.println(“conca=” + strNum + 9);//conca= 1009
```

```
BoxedItem box1 = new BoxedItem("adam", 100,10, true );
Sellable boxi1= new BoxedItem("adamm", 1090,100, true );
System.out.println(boxi1);
box1= (BoxedItem)boxi1;// Down Casting
System.out.println("/n/n");System.out.println(box1);
```

التعميم Generics

• في إطار عمل التعميمات generics framework لاستخدام الأنماط المجردة بطريقة تتجنب العديد من التحويلات الصريحة.
• النوع العمومي generic type هو نوع لايعرف في زمن الترجمة، و إنما 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 بالبارامترات ذات الأنماط الشكلية*، على الرغم من أن أي معرف صالح يمكن أن يستخدم من أجل بارامتر ذو نمط شكلي. • الصيغة العامة لإنشاء كائنات من صنع عام:

```
BaseType <Type> obj = new BaseType <Type>()  // To create an instance (obj )of generic class BaseType
Note: <Type> is classType  as (Integer, Double, Character, ... ) not primitive type as (int, double, char, ...)   ... etc.
```

• 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 Kعلى التوالي. تقوم الطريقة main بإنشاء مثلين من هذا الصنف، الأول من أجل زوج String-Integer والثاني Student-Double: .

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

Using Dialog Boxes

- 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.
  - ✓ showMessageDialog() و showInputDialog() 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 مابين قوسين، كما يلي:

ClassName.methodName(arguments)

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

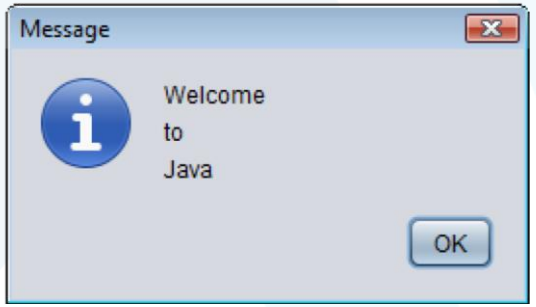
- المنهج showMessageDialog() من الصنف JOptionPane من الحزمة javax.swing له الشكل العام التالي.
- showMessageDialog(arg1,arg2)يعرض مربع حوار العرض نافذة ويتطلب بارامترين،
  - الأول arg1 يساعد تطبيق Javaعلى تحديد مكان وضع مربع الحوار.إذا كانت الوسيطة الأولى null، فسيتم عرض مربع الحوار في وسط الشاشة.
 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

Using Dialog Boxes

- ✓ يسمح صندوق حوار الإدخال input- dialog للمستخدم بإدخال البيانات في البرنامج.
- ✓ يعرض المنهج ()showInputDialog من الصنف JOptionPane مربع حوار الإدخال.
- ✓ يحتوي على مؤشر الادخال 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 الادخال.
- ✓ 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 الى مرجع من النمط String.

Dialog Boxes


```
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,
                              "Welcome\nto\nJava" );
11         } // end main
12    } // end class Dialog1
```



Using Dialog Boxes


```
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

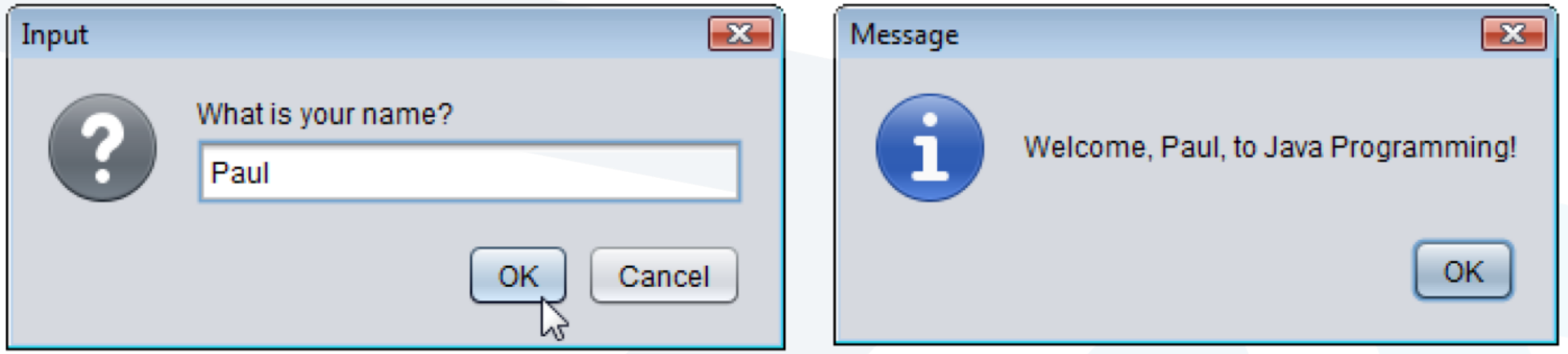


Fig. 3.18 | Obtaining user input from a dialog. (Part 2 of 2.)

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 (java.lang package)، حيث الوسيط args1 هو سلسلة تمثل عددًا صحيحًا وترجع القيمة كعدد نمط int.

4.17 GUI &Graphics

- 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 ولاتتطلب المعرفة الدقيقة بالبرمجة ولغاتها.

Dialog Boxes

3.10 GUI &Graphics

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

Creating Simple Drawings

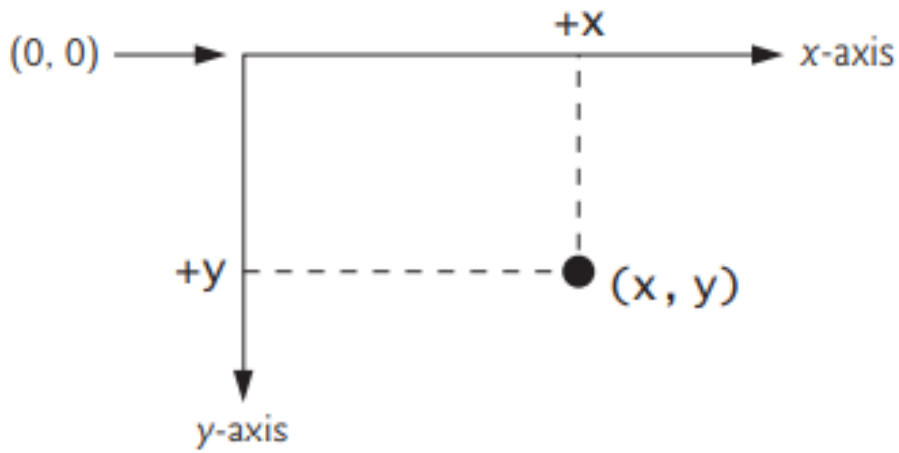
Java's Coordinate System

- ✓ نظام الإحداثيات في Javaعبارة 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 الأفقيhorizontal 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 تسمي awt (Abstract Window Toolkit) والتي تقوم بضبط اعدادات البرنامج تلقائيا حسب المنصة التي يتم تشغيل البرنامج عليها، وهى تنفع في بناء واجهات مستخدم بسيطة، ولكن لا تجدي نفعا في بناء واجهات مستخدم محترفة ومتميزة.
- تم استبدال حزمة awt بحزمة اكثر تميز وكفاءة هي فئة swing وعرفت ب light components weight اي المكونات الخفيفة لإنها تعتمد علي انشاء الكائنات من دون الإعتماد علي منصة التشغيل على خلاف awt والتي تعرف heavy components weight المكونات الثقيلة لانها تعتمد وتتعامل مع منصة التشغيل.
- ومن اجل التفريق بين اصناف حزمة awt واصناف حزمة swing يتم اضافة السابقة J 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 frame window أو أكثر ولكل نافذة اطار شريط عنوان titel bar وحدود border لكي نظهر الإطار نستخدم الصنف JFream من الحزمة javax.swing ويجب:

1- إنشاء كائن من JFream وفق JFrame appli = new JFrame("First");

2- تجديد مقاس الإطار من الطريقة setSize . appli.setSize(300, 300);

3- إضافة الرسمة أو ماتم تجميعة ونرغب بعرضه إلى الإطار appli.add(panel);

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 تحريكة وتغيير حجمه و ... ، وعند إغلاق نافذة الإطار بالضغط على أيقونة الاغلاق من شريط العنونه يظل البرنامج يعمل ولايحدث شيئ سوى إختفاء الإطار، ومن أجل إنهاء البرنامج يجب إستخدام System.exit(0) والتي يجب أن تكون بنهاية main ولكن تخلق مشكلة جديدة وهي ظهور النافذة للحظة وجيزة وينتهي فوراً والرغبة هي انهاء البرنامج عند يضغط 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 انهاء 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`

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

Creating Simple Drawings

4.17 GUI &Graphics

```
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 1 of 2.)

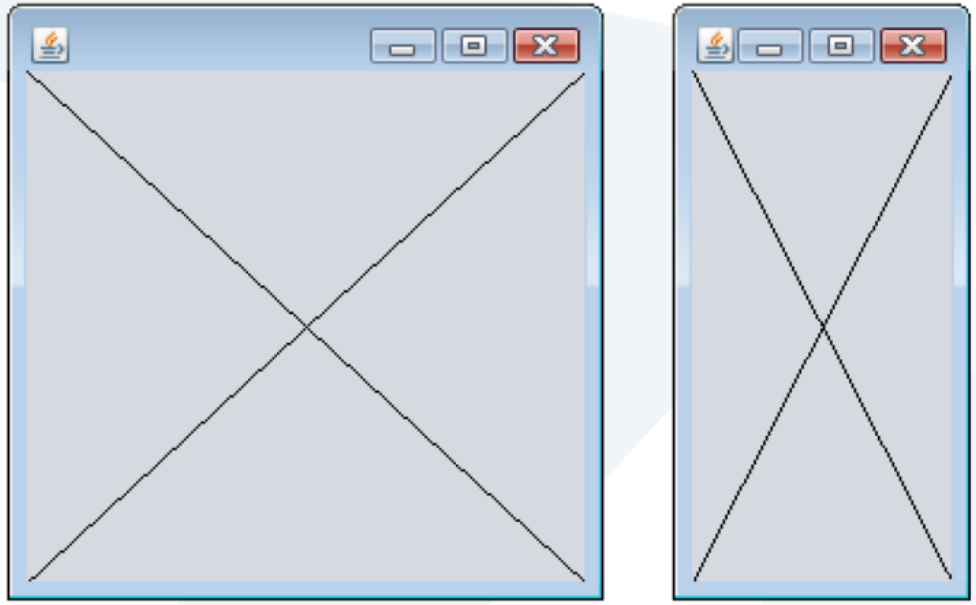


Fig. 4.19 | Creating `JFrame` to display `DrawPanel`. (Part 2 of 2.)

- **class Graphics من الحزمة (java.awt)، والتي توفر طرقًا 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 تلقائيًا 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 تحميلاً 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

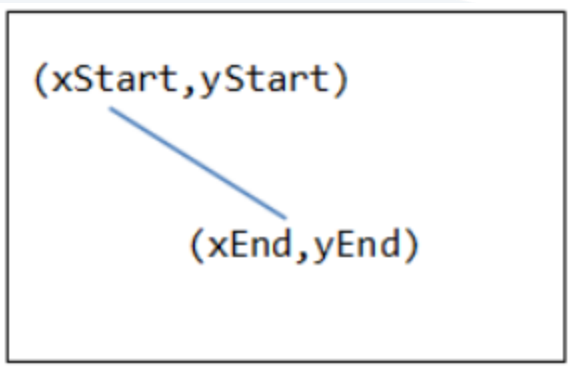
- تقوم أساليب getWidth و getHeight من الصنف JPanel بإرجاع عرض وارتفاع JPanel على التوالي.
- طريقة الرسم drawLine تحتاج أربع وسطاء، أول اثنين هما إحداثيات x و y تعبر عن نقطة البداية نهاية واحدة ، وآخر وسيطتين هما إحداثيات نقطة النهاية الأخرى وترسم خطًا بين نقطتي البداية والنهاية.
- لعرض لوحة الرسم DrawPanel 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 method add من JFrame بإرفاق DrawPanel أو (أي مكون GUIآخر) بإطار JFrame .
- تأخذ method setSize من JFrame معلمتين تمثلان عرض وارتفاع JFrame، على التوالي.
- تعرض method setVisible من JFrame مع الوسيط true الإطار JFrame.
- عندما يتم عرض **JFrame**، يتم استدعاء طريقة **paintComponent** الخاصة بـ **DrawPanel** ضمنيًا.

Drawing Rectangles, Ovals

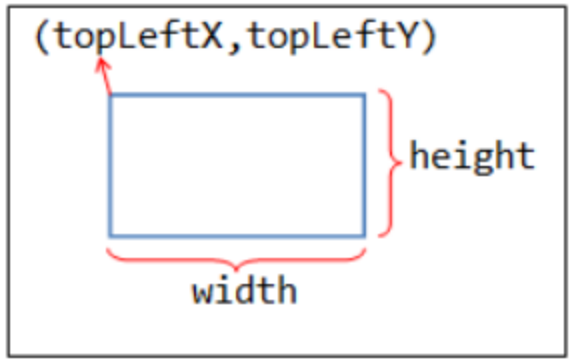
- يتطلب المنهج drawRect أربع وسطاء. يمثل أول اثنان إحداثيات x و y للزاوية اليسارية العلوية للمستطيل ، ويمثل الخياران التاليان عرض المستطيل وارتفاعه.
- البرنامج يرسم القطع الناقص. يقوم بإنشاء مستطيل وهمي يسمى المستطيل المحيط ويضع بداخله شكل بيضاوي يلامس نقاط المنتصف لجميع الجوانب الأربعة.
- يتطلب أسلوب drawOval نفس الوسطاء الأربع 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);
```

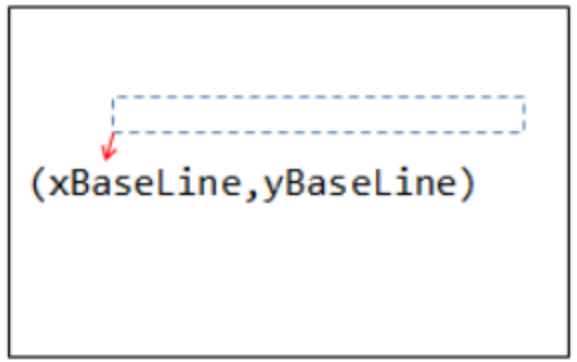
- في JOptionPane، يجب عليك استخدام \n لبدء سطر جديد من النص ، بدلاً من %n ،
- يستخدم منهج parseInt لتحويل السلسلة التي أدخلها المستخدم إلى عدد صحيح ويخزن النتيجة في متغير للنوع الصحسح يتم اختياره.



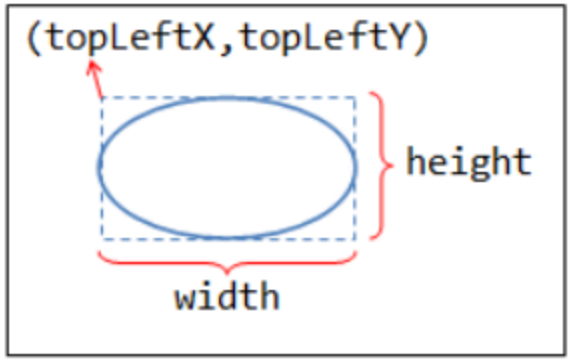
drawLine()



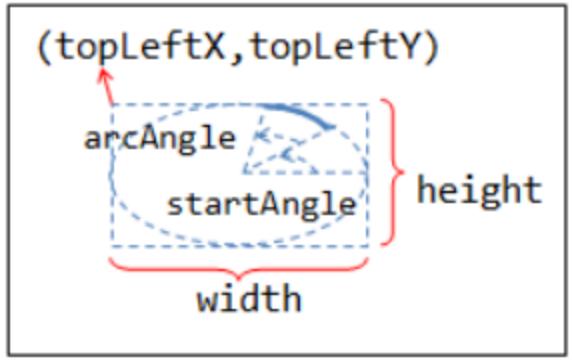
drawRect()



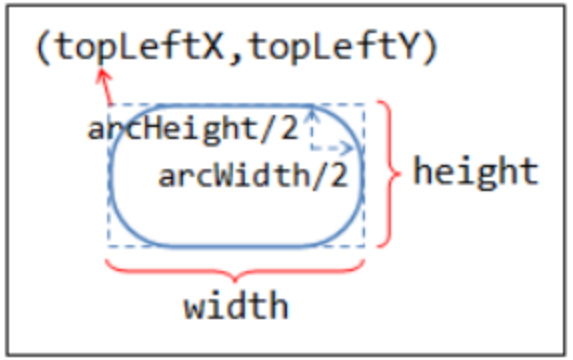
drawString()



drawOval()



drawArc()



drawRoundRect()

Drawing Rectangles and Ovals


```
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 1 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
```

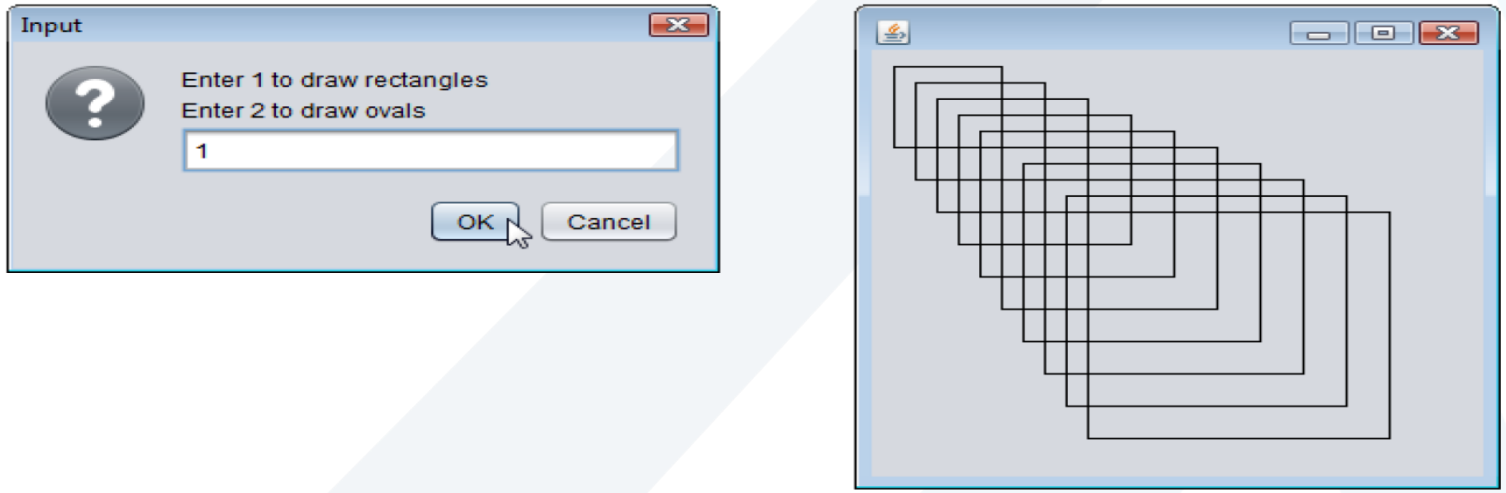


Fig. 5.27 | Obtaining user input and creating a `JFrame` to display `Shapes`. (Part 2 of 3.)

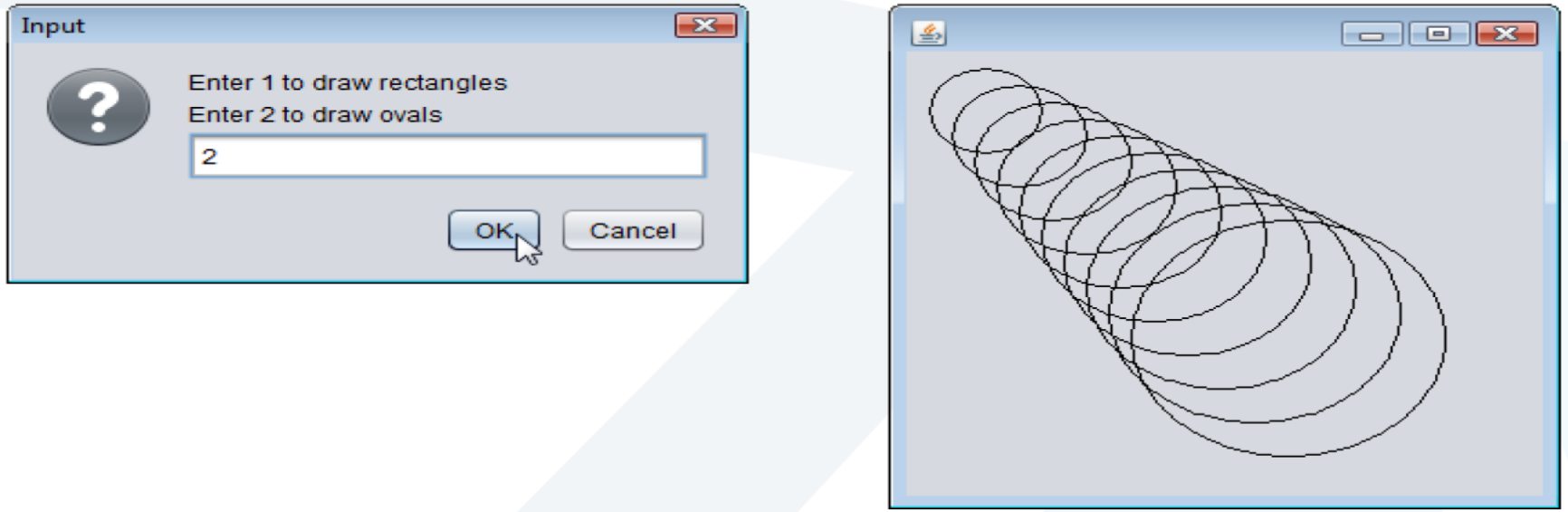


Fig. 5.27 | Obtaining user input and creating a `JFrame` to display `Shapes`. (Part 3 of 3.)

Java's Coordinate System

```
// 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
```

```
        // 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;
```

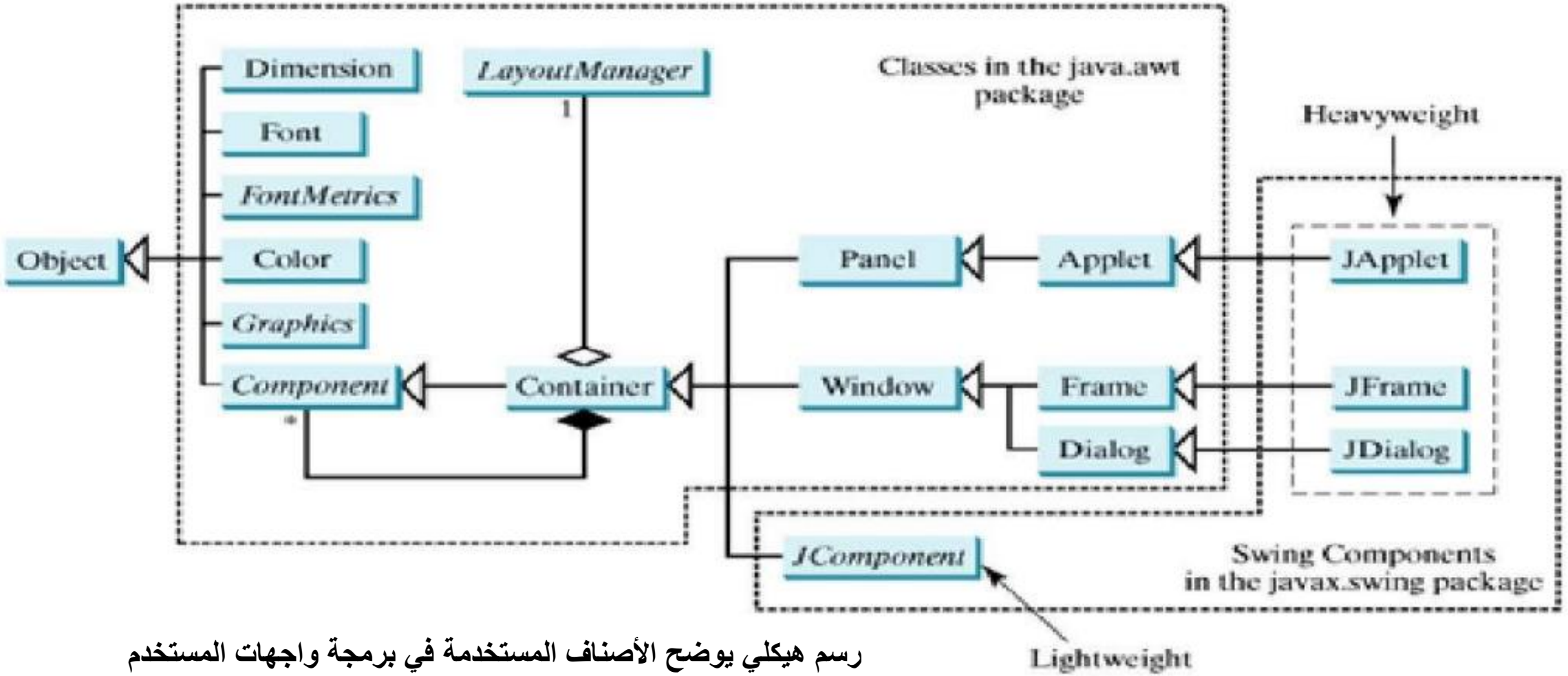
```
case 6:    //draw Fig4.21b Line art with loops and drawLine. P138 9th
        g.drawLine( 10 , 10 + i*25,10 + i * 25, 260);
        g.drawLine( 10 + i*25, 10, 260 , 10 + i * 25);
        g.drawLine( 260 -i*25, 10, 10 , 10 + i * 25);
        g.drawLine( 10+i*25, 260 , 260 ,260 - i*25); break;
case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

```
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
```

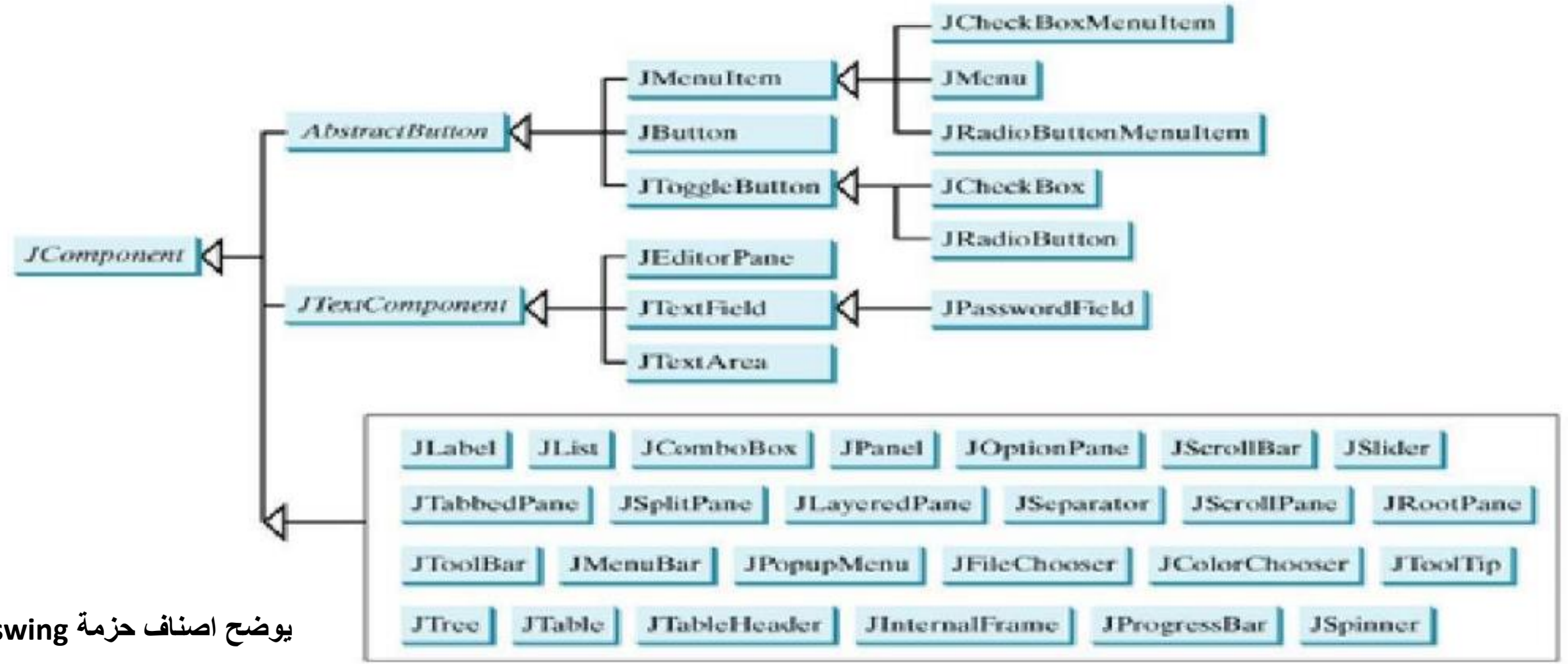
```
// Fig. 5.27 A : ShapesTest0.java
// Test application that displays class Shapes.

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
```

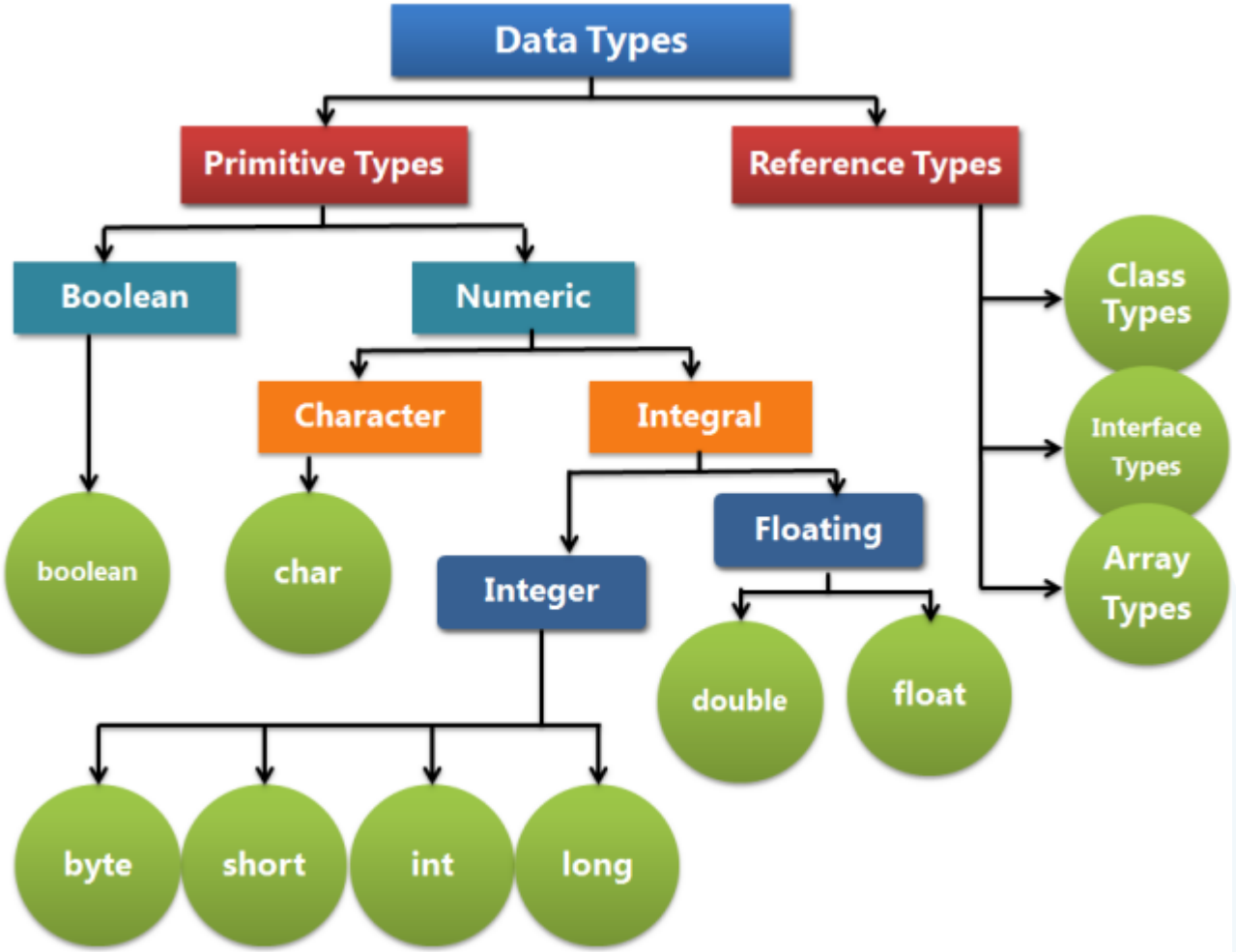
انتهت محاضرة الأسبوع 3

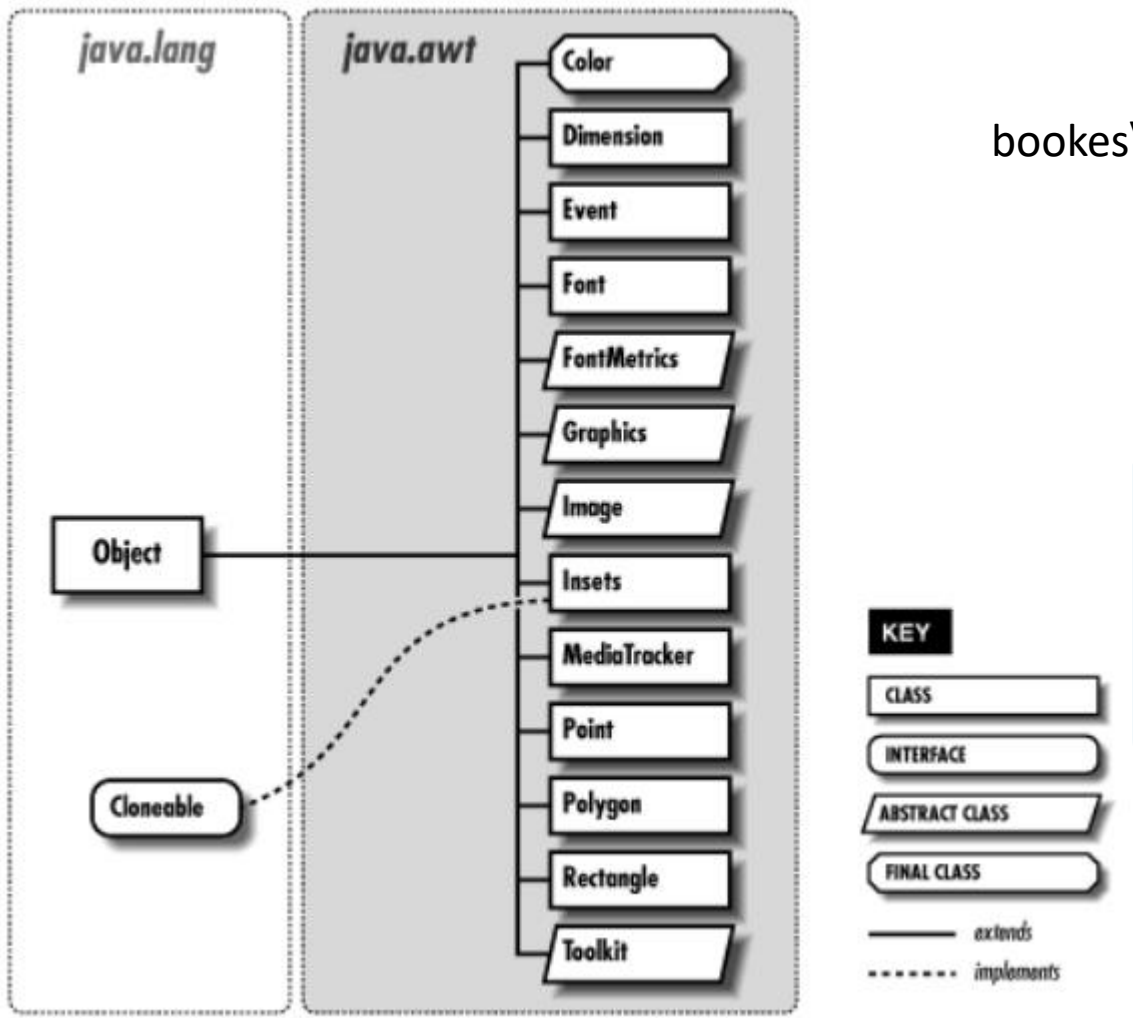


رسم هيكلي يوضح الأصناف المستخدمة في برمجة واجهات المستخدم



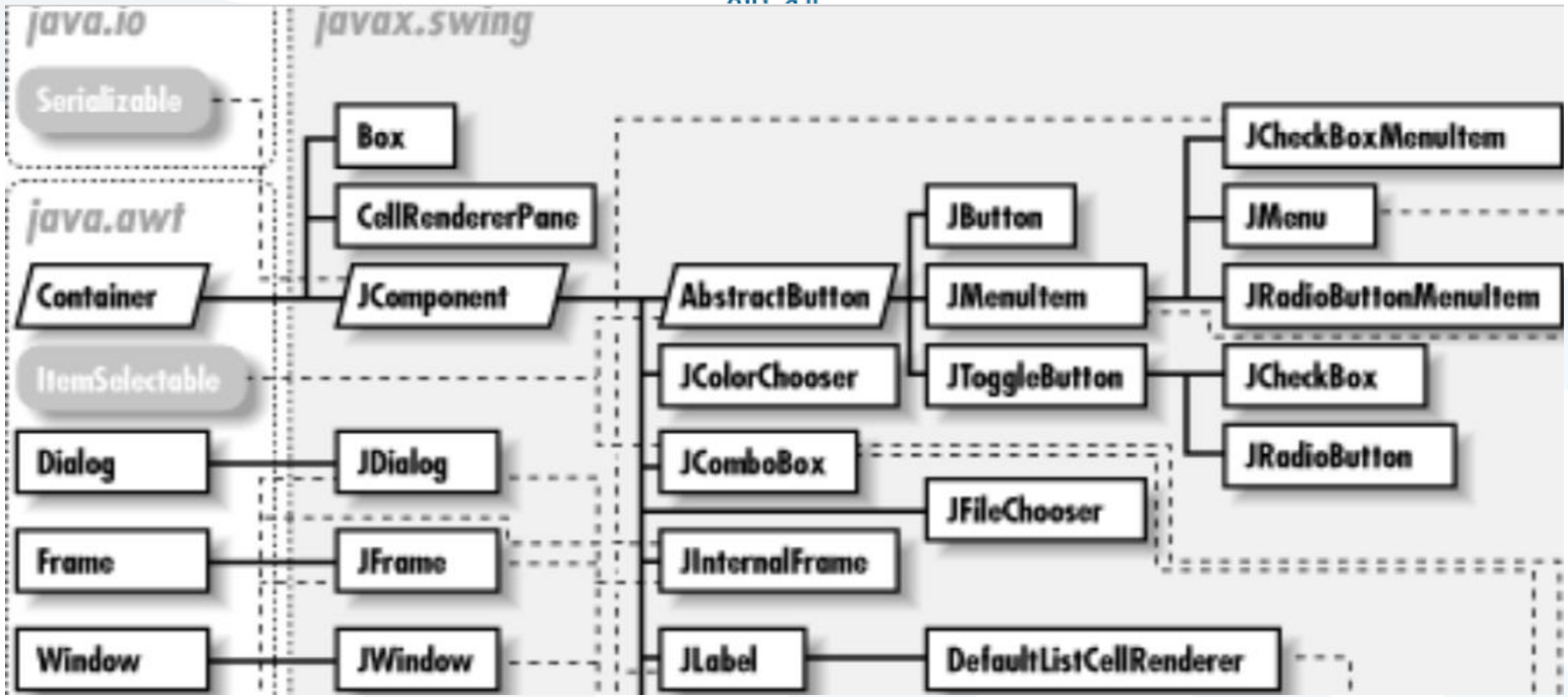
يوضح اصناف حزمة swing

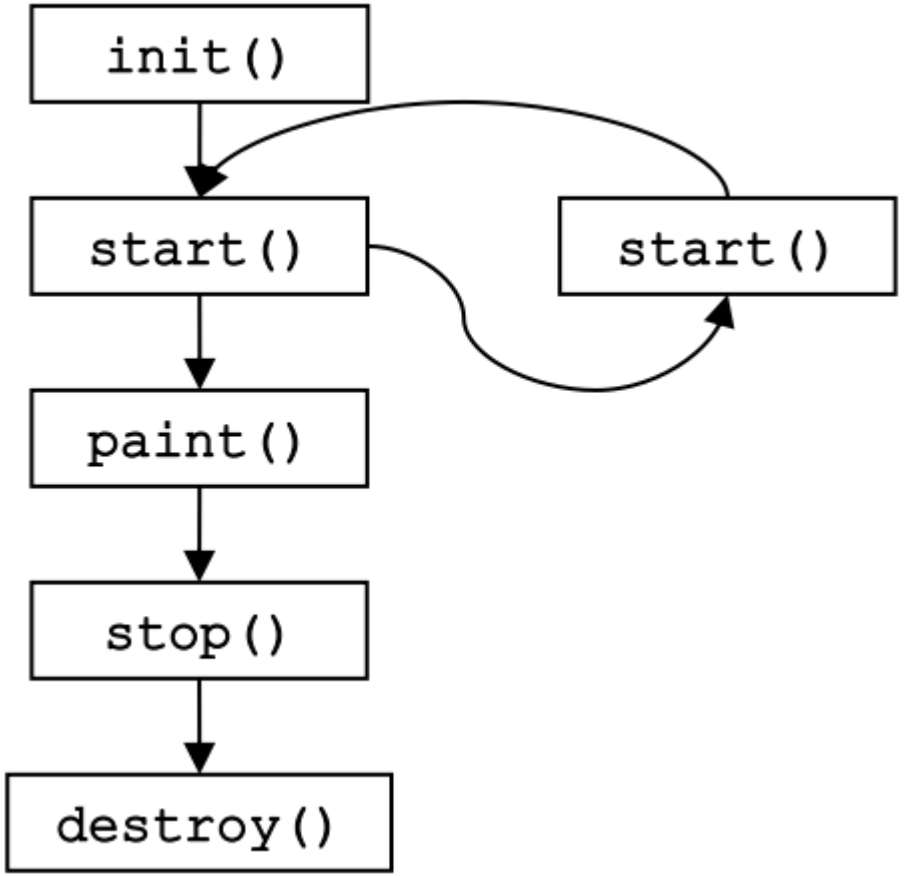

Data Types
Primitive Types
Reference Types
Boolean
Numeric
Class Types
Interface Types
Array Types
Character
Integral
Floating
boolean
char
Integer
double
float
byte
short
int
long



bookes\dataStru\java2021-2022\scrap

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


java.io
javax.swing
Serializable
Box
CellRendererPane
java.awt
Container
JComponent
AbstractButton
JButton
JMenuItem
JRadioButtonMenuItem
JCheckBoxMenuItem
JMenu
ItemSelectable
JColorChooser
JToggleButton
JCheckBox
JRadioButton
Dialog
JDialog
JComboBox
JFileChooser
Frame
JFrame
JInternalFrame
Window
JWindow
JLabel
DefaultListCellRenderer


init()
start()
start()
paint()
stop()
destroy()

colors and filled shapes

Chapter 6.13 p227

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

Graphics methods fillRect and fillOval draw filled rectangles and ovals.