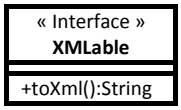
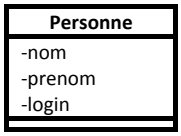
Concept Objet	Exemple	Php5	Java

Interface		interface XMLable {
		  public abstract function toXML( );
		}
			public interface XMLable {
			  public abstract String toXml();
			}

Classe		class Personne {
		  private $nom;
		  private $prenom;
		  private $login;
		}
			public class Personne {
			  private String nom;
			  private String prenom;
			  private String login;
			}

Constructeur		function __construct($nom,
		$prenom, $login) {
		  $this->nom = $nom;
		  $this->prenom = $prenom;
		  $this->login = $login;
		}
			public Personne(String nom,
			String prenom, String login) {
			  this.nom = nom;
			  this.prenom = prenom;
			  this.login = login;
			}

Classe d'implémentation	Personne réalise le contrat défini par XMLable	class Personne implements XMLable {
		  ...
		  public function toXML() {
		  ...
		  }
		}
			public class Personne implements XMLable{
			  ...
			  public String toXml(){
			  ...
			  }
			}

Héritage	PersonneSecure est une Personne disposant d'un mot de passe	class PersonneSecure extends Personne {
		  private $md5pw;
		  ...
		}
			public class PersonneSecure extends Personne {
			  private String md5pw;
			  ...
			}

Redéfinition du constructeur	Le constructeur de PersonneSecure personnalise la construction d'une Personne	class PersonneSecure extends Personne {
		  private $md5pw;
		  function __construct($nom, $prenom,
		  $login, $md5pw) {
		  **parent::__construct($nom, $prenom,
		  $login);**
		  $this->md5pw=$md5pw;
		  }
		}
			public class PersonneSecure extends Personne {
			  private String md5pw;
			  public Personne(String nom, String prenom,
			  String login, String md5pw) {
			  super(nom, prenom, login);
			  this.md5pw=md5pw;
			  }
			}

Redéfinition d'une méthode	La méthode toString est redéfinie dans PersonneSecure (suppose que la classe Personne ait correctement redéfini la méthode toString)	public function __toString() {
		  return parent::__toString() ."$this->md5pw";
		}
			public String toString(){
			  return super.toString() +" " + **this.**md5pw;
			}

Création d'objets (instance)	Création d'une instance de Personne et affichage de son état.	$p = new Personne('Arsac', 'Jacques', 'jarsac'); **echo $p;** **//ou** **echo $p->__toString();**	Personne p = new Personne("Arsac", "Jacques", "jarsac"); System.*out*.println(p.toString()); // ou System.*out*.println(p);

Accès aux attributs public sans les connaître		// $p est une var. objet
		foreach (**$p** as $att => $value) {
		  print "$att => $value\n";
		}
		// voir aussi : get_class_methods
		// get_class_vars
			Utiliser l'introspection en utilisant l'API de reflection.

Exception	Gérer l'exception potentielle	try {
		  ...
		}catch (Exception $e) {
		  ...
		} finally
			try {
			  ...
			} catch (Exception e) {
			  ...
			}**finally**

	Déclencher une exception	throw new MyException('PB');	**throw new** MyException("PB");

Structures "static"	Exemple d'école : un compteur d'instances	class Personne {
		  public static $nbInstances = 0;
		  ...
		}
			class Personne {
			  public static int nbIntances=0;
			  ...
			}

	Accès à un membre	Personne**::**$nbInstances++; ou self::$nbInstances++; en interne	Personne.nbIntances++;
La logique objet est implicitement par référence. La logique implicite des variables non objet est par valeur.	Un même objet peut être référencé par plusieurs variables. Exemple : $p1 et $p2 référencent le même objet.	$p1 = new Personne('Arsac', 'Jacques', 'jarsac'); $p2 = $p1; echo $p1 . " "; echo $p2 . " "; **$p1->setNom**('ARSAC'); echo **$p2** . " "; ce qui donne : Nom : Arsac, Prénom : Jacques Nom : Arsac, Prénom : Jacques Nom : ARSAC, Prénom : Jacques	idem
Constante	Par convention les identificateurs sont en lettres MAJUSCULES.	const MATCH = 1;	**final int MATCH** = 1;